University of the Pacific
Scholarly Commons

University of the Pacific Theses and Dissertations | Graduate School

1973

MANDO : a computer program for symbolic manipulation of differential operators generating continuous transformations

David Kenneth Davison
University of the Pacific

Follow this and additional works at: https://scholarlycommons.pacific.edu/uop_etds

 Part of the Computer Sciences Commons

Recommended Citation

Davison, David Kenneth. (1973). *MANDO : a computer program for symbolic manipulation of differential operators generating continuous transformations*. University of the Pacific, Thesis. https://scholarlycommons.pacific.edu/uop_etds/1806

This Thesis is brought to you for free and open access by the Graduate School at Scholarly Commons. It has been accepted for inclusion in University of the Pacific Theses and Dissertations by an authorized administrator of Scholarly Commons. For more information, please contact mgibney@pacific.edu.

MANDO: A COMPUTER PROGRAM FOR SYMBOLIC MANIPULATION OF DIFFERENTIAL OPERATORS GENERATING CONTINUOUS TRANSFORMATIONS

A Thesis

Presented to

the Graduate Faculty of the

University of the Pacific

In Partial Fulfillment

of the Requirements for the Degree of

Master of Science

---

by

David Kenneth Davison

August 1973

This thesis, written and submitted by

David Kenneth Davison

is approved for recommendation to the Committee on Graduate Studies, University of the Pacific.

Department Chairman or Dean:

Thesis Committee:

Carl E. Wulfman Chairman

Dated Aug 20, 1973

ACKNOWLEDGEMENTS

The author is most grateful to Professor Carl E. Wulfman for his continued interest in the program described herein.

The initial stimulus for the work arose out of the author's study of differential equations with Professor Wulfman and later in Professor Robert L. Anderson's Mathematical Physics course in the spring of 1972. Having worked several examples, the author, by then committed to the field of study, became interested in being able to rapidly test the validity of new results.

With gratefully accepted funding from the National Science Foundation and from the University of the Pacific and support from the Research Corporation, the author developed an instructional derivative program and a primitive operator commutation program which significantly influenced the final form of the program MANDO.

The rest and, in fact, the major part of the development of the program was funded by the Research Corporation, whose contribution is again gratefully acknowledged.

In refining the wording of the first four chapters, I had the help of Professor Wulfman. Dr. John Fletcher, Lawrence Livermore Laboratory, also kindly offered advice on the wording though he was given only short notice.

Finally, I am grateful to my wife, Suzanne, for her understanding and interest.

ABSTRACT

The program MANDO efficiently performs computations involving pairs of operators, a single operator, and operators applied to functions, saving time and cost over pencil-and-paper methods.

A versatile but compact data structure, defined under SPITBOL's facility for creation of datatypes, contains the operators (and functions) and provides a means for systematically referencing their relevant parts.

On input, functions and operators are written in a restricted but natural string format, for which the program can readily convert them to the internal data structure.

Central to the method of operation of the derivative routine is its ability to differentiate a function written as a string. This allows for a certain compactness in the internal form. To counteract the relative slowness of string processing in the derivative routine, the program keeps a table of derivatives repeated during the processing. The table is checked for the presence of the function and its derivative before the derivative sequence is applied.

Some simplification is performed. The simplification relies on the ordering sequence, followed by a sequence which cancels or combines terms that are alike, except, in general, for numerical multipliers.

TABLE OF CONTENTS

Introduction

When it is desired to know the results of operations performed with the differential operators that are generators of the Lie groups of differential equations derived by Lie's (1)(2) or other methods (3), a prodigous amount of time may be required when the work is done by pencil-and-paper calculation. In addition, the results are often erroneous due to the presence of errors which are not detected during the process of calculation. This thesis reports work which has been directed towards the end of improving the reliability of and reducing the time required in evaluating the effects of differential operators. The tools are the Stanford IBM 360/67 computer, the local terminal of the same, and a program MANDO (for MANipulation of Differential Operators) written in SPITBOL (4), a sublanguage of SNOBOL (5).

In Chapters I, II and III we examine the capabilities, the format, and the input method for the program. With these we will be able to write instructions with which the program may carry out the types of calculations reviewed in Chapter I. In Chapter IV we see the application of the program to several problems.

We review the data structures, the cannonical ordering sequence, and the details of the program function ("p-function") operations in Chapter V. Chapter VI deals with the methods of changing the program to suit applications other than those for which it is currently intended.

Chapter I

Program Capabilities

Given differential operators A and B, the program is capable of evaluating simple products, commutators AB - BA = [A,B], and anti-commutators AB + BA. Using the commutator option, one can quickly determine the structure of a set of operators which are generators of a continuous group. Around two of the aforementioned capabilities are also built the recursive operations AB and [A,B], starting with B $\rightarrow$ AB or [A,B]. Thus information about the operator created by the similarity transformation $e^{aS}Xe^{-aS} = X + a[S,X] + \frac{a^2[S,[S,X]]}{2!} + \ldots$*, where X and S are operators, may be obtained.

Since the program is also capable of evaluating the application of an operator to a function f, information about finite transformations may also be obtained (the finite transformation of function f corresponding to operator X is $e^{aX}f = \left\{1 + aX + \frac{(aX)^2}{2!} + \ldots\right\}f$**).

There are limitations to the types of operators the program will process. Besides the format limitations and the limitations on the types of functions (exp, sin, cos, lgn and sqt) within the format, which will be discussed later (Chapter II, p. 5), there are restric-

* See Campbell (6), pp. 90 and 91, for linear operators.

** ibid., p. 41

tions on the orders* of the two operators which comprise the product, commutator or anticommutator. These restrictions stem from the consideration that the resultant operator can have an order no higher than 9; additionally, neither operator of a pair can have an order higher than 9. For operators Q_1 and Q_2 with orders n_1 and n_2, the program will not process if $n_1 > 9$ or $n_2 > 9$. It will, in

	Term	
Order	Finite Transformation	Similarity Transformation
0	1	-
1	aX	X
2	$\frac{(aX)^2}{2!}$	a[S,X]
3	$\frac{(aX)^3}{3!}$	$\frac{a^2[S,[S,X]]}{2!}$
⋮	⋮	⋮

Table 1.1 Indexing of terms in series expansions of finite and similarity transformations.

* The order of an operator is the largest number of derivative coefficients associated with any additive component of the operator, e. g., the orders of the operators $x\partial_x$ ($\partial_x = \frac{\partial}{\partial x}$) and $-3gx^3\partial_r\partial_v$ + $2vux\partial_u\partial_v\partial_x$ are, respectively, 1 and 3.

general, produce errors in the derivative coefficients of commutator results with $n_1 + n_2 > 10$ and of product results with $n_1 + n_2 > 9$. Otherwise the results are reliable. Operators of order zero are permitted.

In spite of the limitations described, the program is very useful in the evaluation of finite and similarity transformations. Labelling the orders of the terms in the series expansions of these transformations by the method illustrated in Table 1.1, the limitations of the analyses can be expediently described by Table 1.2.

	Limiting Order in Series					
	Finite Transformation	Similarity Transformation				
		Order of S				
Order of X		0	1	2	3	4
0	∞	1*	∞	10	5	4
1	9	2*	∞	9	5	3
2	4	3*	∞	8	4	3
3	3	4*	∞	7	4	3
4	2	5*	∞	6	3	2
5	1	6*	∞	5	3	2

Table 1.2 Tabulation of limiting order in series when finite and similarity transformations are being studied. Further entries for similarity transformations may be found by noting that an entry is the largest integer less than $\frac{10 - n}{m - 1} + 1$, where n is the order of X and m, the order of S, subject to $n + m \leq 10$. The asterisk (*) indicates that further evaluation will be of commutators of zeroth-order operators, which are always zero.

Chapter II

Function and Operator Format

The format chosen for communication with the computer is a compromise between a style which can be easily read by the user and a style which can be easily interpreted by the computer. The format is nonstandard; the arguments appearing in Appendix A are accordingly presented in its defense. The chart at the end of this section (pp. 9 and 10) summarizes the discussion and may be useful in learning the format. In the following, the symbols and expressions enclosed in quotes ("...") are examples of forms which may be put into the computer.

The arithmetical symbols used are: "+" (addition), "-" (subtraction), "@" (exponentiation), and "/" (division). No symbol is needed for multiplication (If "*" is used, it is removed before evaluation.). The arguments to "@" and "/" must be enclosed in parentheses: x^2 is written "X@(2)", and $\frac{1}{1 - x^2}$ is written "1/(1-X@(2))".

Some useful functions* available are: "EXP" (exponential), "SQT" (square root), "SIN" (sine), "COS" (cosine), and "LGN" (logarithmec). Parentheses around the arguments to these are also required: $\sin^2 x$ is written "SIN(X)@(2)".

* On p. 126 is a technical description of how to add other useful functions to the set of those available.

Variables may be chosen from: "R", "T", "U", "V", "W", "X", "Y" and "Z".

Constants may be chosen from "A", "B", "F", "G", "H", "J", "K", "M", "N", "O", "P", and "Q" or any of these letters followed by an integer composed of the digits: "1", "2", "3", "4" and "5", which may be thought of as subscripts; "A", "A1" and "A23" are, for example, recognized as distinct constants. Besides the constants themselves, powers and sums of constants may appear as factors in an additive component of a function: $ab^2x + a(a - ab)y^2$ is written "AB@(2)X+A(A-AB)Y@(2)". The powers must be real numbers or integers: $b^{4/3}$ is "B@(1.33)" and b^2 is "B@(2)". When writing third and fourth roots, one includes two places past the decimal point; for square roots, only one place. Sums may be written as sums of integer or real number multiples of the constants, e. g., "(-JK+2.5G2-4)"; these sums may not contain powers or other sums of constants as factors in the additive components but may contain products of constants: "H(2J@(2)-(HJ-1)K)X@(2)" is not acceptable, but "H(2JJ-HJK+K)X@(2)" is. Combinations of these forms are acceptable: $ab^{4/3}(-jk + 5g_2/2 - 4)n_{23}$ is "AB@(1.33)(-JK+2.5G2-4)N23".

The letter "I" is interpreted to be a constant, but it is also treated as being equal to $\sqrt{-1}$.

Functions are written as sums of additive components. Each component is composed of five elements: (1) "+", "-" or nothing (null); (2) any real number or integer; (3) constant multipliers; (4) functions of the variables; and (5) divisors. A typical function is: "2IEXP(IT)(1-X@(2))+1/(R(1-X@(2)@(.5))", equivalent to

$2ie^{it}(1 - x^2) + \frac{1}{r\sqrt{1 - x^2}}$. The elements of the additive components of this function are classified as follows:

Additive Component	Element				
	1	2	3	4	5
1	"-"	"2"	"I"	"EXP(IT)(1-X@(2))"	-
2	"+"	"1"	-	-	"R(1-X@(2))@(.5)"

Elements of other functions may be similarly classified.

The arguments to the "useful functions" "EXP", "SQT", "SIN", "COS" and "LGN", divisors, and parenthesized factors have the same format as functions in general; That is, they are written as sums of additive components composed of the five elements described above (except for divisors, which may not contain divisors themselves: before $g \cdot \sin(1 - 1/x)/(1 - 1/x)$ is translated, it is first changed to $gx \cdot \sin(1 - 1/x)/(x - 1)$, then written as "GXSIN(1-1/(X))/(X-1)").

Certain forms will result in the detection of errors: those in which the parentheses do not close, e. g., "SIN(2X"; those which contain powers which are functions of the variable, e. g., "2X@(2X+6)"; those which contain powers of powers, e. g., "X@(2)@(3)"; and those containing divisors of the form "/(" argument ")(" argument ")". Divisors containing divisors, e. g., "1/(1/(X))", cause errors also; since divisors may appear in the arguments of the "useful functions" and in the additive components of parenthesized factors, they may appear, however, within these forms in the divisor as a whole, e. g., "1/(SIN(1/(X)))" is acceptable. If such forms are detected late in the execution, their derivatives may be assigned the value "$$" (undefined derivative), and they will appear on the printout as such.

Powers of variables, "useful functions", and parenthesized

factors may be constant expressions composed of sums of real or integer multiples of constants, e. g., "X@(2.5G-6)" or "COS(X@(2))@(2JN-M+6)". If one of the terms is a number ("6" in the examples), it is written as the last element in the sum, as shown.

Differential operators are constructed by attaching sets of derivative coefficients to the right ends of components which have the same form as additive components of functions. Differential operators have a form very similar to that of "functions composed of additive components", e. g., the operator $kx\partial_x + (x^2 + y^2)\partial_y$ is written "KXDX+(X@(2)+Y@(2))DY". Within the format a set of derivative coefficients is indicated by alternating the letter "D" with the variables of differentiation: $\partial_x\partial_u\partial_u$ is written "DXDUDU". Writing, e. g., sets such as "DADX", "DDX" or "DXX" results in rejection of an input before any extensive processing has been performed.

The operator $\frac{i}{\sqrt{1 - x^2}}\,\partial_r\partial_v + \frac{i}{2\sqrt{1 - x^2}}\,\partial_v$ may be written "I/((1-X@(2))@(.5))DRDV+.5I/((1-X@(2))@(.5))DV". Operators such as this with factors common to each term may be shortened as follows: COMMON FACTOR "|(" OPERATOR WITH COMMON FACTOR REMOVED ")" (The vertical bar "|" is a standard symbol on many computer keyboards.); accordingly: "I/((1-X@(2))@(.5))|(DRDV+.5DV)". Another and, in fact, a preferable way to write the same operator is "I(1-X@(2))@(-.5)|(DRDV+.5DV)" (the technical aspects of the preference for using negative exponents rather than divisors are given on p. 91).

The following chart summarizes the description of format given in Chapter II:

FORMAT CHART

Arithmetical Operators

Addition	Subtraction	Multiplication	Division	Exponentiation
"+"	"-"	(none)	"/("...")"	"@("...")"

Useful Functions Available

"EXP("...")", "SQT("...")", "SIN("...")", "COS("...")" and "LGN("...")"

Variables

"R", "T", "U", "V", "W", "X", "Y" and "Z"

Constants

Letters "A", "B", "F", "G", "H", "J", "K", "M", "N", "O", "P" or "Q", alone or followed by an integer composed of the digits "1", "2", "3", "4" and "5", e. g., "A", "A1" and "A23". Real or integer powers are permitted. The letter "I" is interpreted as $\sqrt{-1}$.

Elements of Additive Components

Element	Contents
1	"+", "-" or null
2	integer or real number
3	constants
4	functions of the variables
5	divisors

Divisors, arguments to the "useful functions", and parenthesized factors have the same form as functions in general; divisors may contain only the first four elements above. Powers of functions may be sums of real or integer multiples of constants.

Forms Which Cause Errors in Execution

Parentheses not closed: "SIN(2X"

Power which is a function of the variable: "2X@(2X+6)"

Power of a power: "X@(2)@(3)"

Divisor of form: "/("...")("...")"

Divisors containing divisors: "1/(1/(X))"

FORMAT CHART
(continued)

Derivative Coefficients

Derivative coefficients are formed by alternating the letter "D" with the variables: "DXDUDU"

Operators with a Common Factor in Each Term

The form ..."|("...")" may be used.

Chapter III

Method of Input

Once function and operator formats are known, the next step in making the program work is the establishment of the manner in which instructions are passed on to the processing unit during the time the program is loaded into the computer. There are two ways to run a program, interactive mode and batch mode. The former permits direct input from the terminal at each step. Speed of execution is, however, sacrificed. In batch mode the entire set of processing steps (instructions and data input) are figured out ahead of time, and the computer is allowed to find a time when it can apply the full power of its facilities to the problem presented. The manner of setting up a batch mode execution follows; after this are sample runs in interactive and batch modes to illustrate the timing problem.

There are eight four-letter instructions which control the flow of execution, and there is an exact way in which an operator pair is to be made available to the computer for manipulation. The input is included in the first 80 columns (there are 132 columns on the keyboard). Since some operators are longer than 80 characters, provision is made for them to be input on several lines, if necessary. An operator pair is written as follows: OPERATOR "#" OPERATOR ":". The pair is broken into parts (never in the middle

of a constant) which fit on a line and is fed in until the colon is reached. This is the first step in the execution. The next step is the specification of the procedure. This is done on a new line. If only the commutator, anticommutator or product is desired, the instruction "COMM", "ANTI" or "PROD" is used. Thus the sequence:

```
"XD X # DX  :"          (1)
"  COMM"                (2)
"TERM"                  (3)
```

results in: (1) the operator pair $x\partial_x$, ∂_x being made available for manipulation, (2) the commutator being taken and displayed, and (3) termination of the program ("TERM" terminates the program). Note that blanks are removed from input expressions. If "CHEK" had been used in place of "COMM", a system dump would have resulted which could be examined to determine whether the operator pair had been properly written. Before using "CHEK" it is advisable to know the material in Chapter V. The instruction "APLY" followed by a function f could also have been used in place of "COMM". This would have caused the application of the second operator (∂_x in this case) to the function f. Using "APLY" in this manner requires that the first operator be some dummy value, the simpler the better ("1" would be good). When "APLY" is used after "SAVE", as explained in the next paragraph, the first operator is no longer a dummy value.

If an operator pair is followed by the instruction "SAVE" (on a new line), the computer will save the results of further computations and prompt for additional inputs. "SAVE" is followed by one

of the instructions "COMM", "ANTI", "PROD" or "APLY". Each of these and only these bring about a display of a result (the respective commutator, anticommutator, product, or transformed function). The other four instructions, "TERM", "CHEK", "SAVE" and "OPER", are for control and checking. "OPER" ends the action of "SAVE" and causes a prompt for another operator pair. While "SAVE" is in effect, the result arising from a "COMM", "ANTI" or "PROD" instruction becomes the second operator of the pair for the next operation. Assuming Q_1, Q_2 and Q_3 to be explicit operators and f_1 and f_2 to be explicit functions, the following illustrates the use of the "SAVE" instruction:

Input	Action
Q_1 "#" Q_2 ":"	operator pair Q_1, Q_2 is made available for manipulation
"SAVE"	the program saves the results of further instructions; the result becomes the second operator (except for the result of "APLY", which is never retained)
"COMM"	$[Q_1,Q_2]$ is determined and displayed; operator pair Q_1, $[Q_1,Q_2]$ is available for manipulation
"PROD"	$Q_1[Q_1,Q_2]$ is determined and displayed; operator pair Q_1, $Q_1[Q_1,Q_2]$ is available for manipulation
"COMM"	$[Q_1,Q_1[Q_1,Q_2]]$ is determined and displayed; operator pair Q_1, $[Q_1,Q_1[Q_1,Q_2]]$ is available for manipulation
"OPER"	the previous result is lost, and the program is prepared for another pair of operators
Q_3 "#" Q_3 ":"	operator pair Q_3, Q_3 is made available for manipulation
"SAVE"	(see earlier reference)

"PROD"	Q_3Q_3 is determined and displayed; operator pair Q_3, Q_3Q_3 is available for manipulation
"APLY" f_1	application of operator Q_3Q_3 to f_1 is determined and displayed
"APLY" f_2	application of operator Q_3Q_3 to f_2 is determined and displayed
"PROD"	$Q_3Q_3Q_3$ is determined and displayed; operator pair Q_3, $Q_3Q_3Q_3$ is available for manipulation
"APLY" f_1	application of operator $Q_3Q_3Q_3$ to f_1 is determined and displayed
"OPER"	(see earlier reference)
Q_1 "#" Q_3 ":"	operator pair Q_1, Q_3 is made available for manipulation
"COMM"	$[Q_1,Q_3]$ is determined and displayed
Q_2 "#" Q_3 ":"	operator pair Q_2, Q_3 is made available for manipulation
"COMM"	$[Q_2,Q_3]$ is determined and displayed
"TERM"	program termination
"TERM"	(explained below)

Notice that, after "OPER" has ended the action of "SAVE", another operator pair may be given. When "SAVE" is not in effect, another operator pair may be given after "COMM", "ANTI", "PROD", "APLY", or "CHEK". Whenever an operator pair may be given, a "TERM" instruction may be given. When "TERM" is given while "SAVE" is in effect, an error is detected; if the next input is also "TERM", the program terminates. Thus it is wise to end a set of inputs with two "TERM" instructions, as is shown, to assure normal termination. Table 3.1, p. 15, summarizes the use of the four-letter instructions. Figure

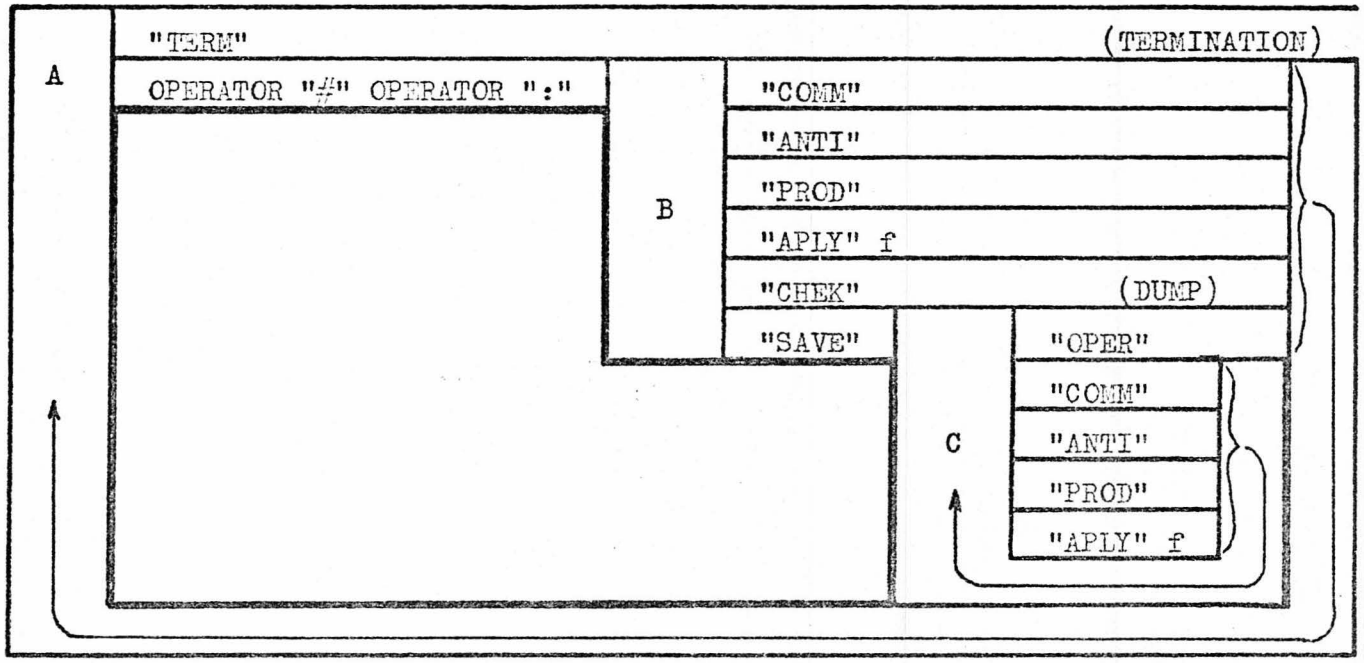

Table 3.1 Chart of acceptable input sequences. Failure to respond in the required manner results in a return to position A.

5.3.3, p. 122, a flowchart for the entire program, gives a similar overview.

On the following two pages is an example of the comparison of the time required to execute the commutators of operators $\exp(g_1x - g_2x) \cdot v\partial_u$ and $u\partial_u$ and of $\exp(g_1x - g_2x) \cdot v\partial_u$ and $v\partial_v$ in both interactive and batch modes. It is noticed that before each input there is a prompt and that the passage of time is recorded in some detail. The former serve as reminders of the meanings of the input information. The latter permit accurate estimates of efficiency and cost. Interactive processing is inherently more time-consuming and costly, as is clearly illustrated by the example.

(INTERACTIVE)

```
? snobol
:ENTERING STANFORD/SNOBOL
? core=44
? go

COMPILATION TIME    8.411 SECONDS
MEMORY USAGE (DECIMAL BYTES)
CODE:              93670
STRINGS:            4769
VARIABLES:         14464
CONSTANTS:          3800
TOTAL:            116703
AVAILABLE:         58740

* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  8991 MSEC, COUNT:  71 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
exp(g1x-g2x)vdu # udu :
EXP(G1X-G2X)VDU # UDU :
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (9741 MSEC, 249 STATEMENTS)
comm
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  10167 MSEC, 329 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  10242 MSEC, 367 STATEMENTS.

 ORDERED RESULT:


 +< EXP(G1X-G2X)V >DU


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  10443 MSEC, 391 STATEMENTS.
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  10910 MSEC, 446 STATEMENTS.

 SIMPLIFIED RESULT:


 +< VEXP(G1X-G2X) >DU



* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  11263 MSEC, COUNT:  475 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
exp(g1x-g2x)vdu # vdv :
EXP(G1X-G2X)VDU # VDV :
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (12413 MSEC, 637 STATEMENTS)
comm
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  12709 MSEC, 729 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  12816 MSEC, 765 STATEMENTS.

 ORDERED RESULT:


 +< -VEXP(G1X-G2X) >DU


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  13038 MSEC, 789 STATEMENTS.
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  13110 MSEC, 810 STATEMENTS.

 SIMPLIFIED RESULT:


 +< -VEXP(G1X-G2X) >DU



* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  13594 MSEC, COUNT:  839 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
term
TERM
NORMAL TERMINATION
NUMBER OF STATEMENTS EXECUTED          843
EXECUTION TIME (SECONDS)            13.712
MSEC/STATEMENT                      16.265
NUMBER OF STORAGE REGENERATIONS          7
? exit
:EXITING STANFORD/SNOBOL
?
```

```
COMPILATION TIME    4.478 SECONDS

MEMORY USAGE (DECIMAL BYTES)

CODE:              93682

STRINGS:            4769

VARIABLES:         14464

CONSTANTS:          3800

TOTAL:            116715

AVAILABLE:         58721

SUCCESSFUL COMPILATION
```

(BATCH)

```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  48 MSEC, COUNT:  71 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
EXP(G1X-G2X)VDU # UDU :
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (276 MSEC, 249 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  300 MSEC, 329 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  306 MSEC, 367 STATEMENTS.

 ORDERED RESULT:

 +< EXP(G1X-G2X)V >DU


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  318 MSEC, 391 STATEMENTS.
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  446 MSEC, 446 STATEMENTS.

 SIMPLIFIED RESULT:

 +< VEXP(G1X-G2X) >DU


* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  466 MSEC, COUNT:  475 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
EXP(G1X-G2X)VDU # VDV :
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (722 MSEC, 637 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  762 MSEC, 729 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  768 MSEC, 765 STATEMENTS.

 ORDERED RESULT:

 +< -VEXP(G1X-G2X) >DU


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  781 MSEC, 789 STATEMENTS.
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  784 MSEC, 810 STATEMENTS.

 SIMPLIFIED RESULT:

 +< -VEXP(G1X-G2X) >DU


* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  804 MSEC, COUNT:  839 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
TERM


NORMAL TERMINATION IN STATEMENT      1020

NUMBER OF STATEMENTS EXECUTED         843

EXECUTION TIME (SECONDS)            0.812

MSEC/STATEMENT                      0.963

NUMBER OF STORAGE REGENERATIONS         7
```

When operators are put into the computer, they are examined for the presence of variables, constants, and functions. Except for the variables, the order in which these factors appear on input is the same as the order in which they will appear on output (after simplification); variables, on the other hand, are the reverse of their order on input. Before any operators are introduced, variables, constants, and functions may be written in the order which will produce the desired result on output, followed by "#:", and the ordering sequence will be cued. In the example on the next page, the input "WVUX#:" sets up the ordering so that the "simplified result" is ordered "XUVW". The order in which the derivative coefficients are printed out is also affected by this cueing. The technical details are found in Section 5.2, pp. 89 to 92.

```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  44 MSEC, COUNT:  73 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
WVUX#:
 NULL OPERATOR (2)
 INPUT REJECTED.


* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  102 MSEC, COUNT:  168 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
(3X@(2)V -2X@(3)W)DX  +4U@(2)DU  +(8UV -6XV@(2) +3X@(2)VW)DV
+(2V@(2) +8UW -12XVW +6X@(2)W@(2))DW
#
(XV -X@(2)W)DX  +(-V@(2) +XVW)DV  +(-2VW +2XW@(2))DW
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (831 MSEC, 1078 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  1531 MSEC, 3919 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  1660 MSEC, 4733 STATEMENTS.

 ORDERED RESULT:


 +< 3X@(2)VV -2X@(3)WV -6X@(2)VWX +4X@(3)WWX -6XVVX +6X@(2)WVX +6XVWX@(2) -6X@(2)WWX@(2)
 +3V@(2)X@(2) -3XVWX@(2) +8UVX -6XV@(2)X +3X@(2)VWX -4VWX@(3) +4XW@(2)X@(3) -2V@(2)X@(2)
 -8UWX@(2) +12XVWX@(2) -6X@(2)W@(2)X@(2) >DX

 +< 3X@(2)VVW -2X@(3)WVW +6XVV@(2) -6X@(2)WV@(2) -6XVVWX +6X@(2)WVWX +8V@(2)U -8XVWU -12V@(2)XV
 +12XVWXV +3V@(2)X@(2)W -3XVWX@(2)W -16UVV +12XV@(2)V -6X@(2)VWV +8UVXW -6XV@(2)XW
 +3X@(2)VWXW +2V@(2)XV +8UWXV -12XVWXV +6X@(2)W@(2)XV +6VWX@(2)V -6XW@(2)X@(2)V >DV

 +< 6X@(2)VW@(2) -4X@(3)WW@(2) +12XVVW -12X@(2)WVW -12XVW@(2)X +12X@(2)WW@(2)X +4V@(2)V
 -4XVWV -12V@(2)XW +12XVWXW -16UVW +12XV@(2)W -6X@(2)VWW -4V@(2)V -16UWV +24XVWV -12X@(2)W@(2)V
 +8V@(2)XW +32UWXW -48XVWXW +24X@(2)W@(2)XW +16VWU -16XW@(2)U -24VWXV +24XW@(2)XV +24VWX@(2)W
 -24XW@(2)X@(2)W >DW



 TIME AND COUNT AT END OF OUTPUT ROUTINE:  2005 MSEC, 5304 STATEMENTS.
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  3509 MSEC, 7941 STATEMENTS.

 SIMPLIFIED RESULT:


 +< -8X@(2)V@(2) +12X@(3)VW -4X@(4)W@(2) +8XUV -8X@(2)UW >DX

 +< -12X@(2)V@(2)W +4X@(3)VW@(2) +8XV@(3) +8XUVW -8UV@(2) >DV

 +< -24X@(2)VW@(2) +8X@(3)W@(3) +16XV@(2)W +16XUW@(2) -16UVW >DW



* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
```

Chapter IV

Application to Typical Problems

Of the examples in this chapter, numbers 2, 3 and 4 were run on an essentially identical but slower version of MANDO. The timing of Examples 1, 5 and 6 is therefore more typical of the results that may be expected. The thesis would be, of course, more complete if all the examples were run using the same version, but the funds have been restricted for this work.

Example 1. We begin by studying the generators of the projective group of the plane, which has a structure easy to verify; thus we will be able to concentrate more on the use of the program than on the problem posed. For $Q_1 = \partial_x$, $Q_2 = \partial_y$, $Q_3 = y\partial_x$, $Q_4 = x\partial_y$, $Q_5 = x\partial_x$, $Q_6 = y\partial_y$, $Q_7 = x(x\partial_x + y\partial_y)$, and $Q_8 = y(x\partial_x + y\partial_y)$, the commutator table is:

$[\rightarrow, \downarrow]$	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7
Q_2	0	-					
Q_3	0	Q_1	-				
Q_4	Q_2	0	$-Q_5+Q_6$	-			
Q_5	Q_1	0	Q_3	$-Q_4$	-		
Q_6	0	Q_2	$-Q_3$	Q_4	0	-	
Q_7	$2Q_5+Q_6$	Q_4	Q_8	0	Q_7	0	-
Q_8	Q_3	Q_5+2Q_6	0	Q_7	0	Q_8	0

The 28 commutators indicated may be calculated by giving the program MANDO sets of operators and instructions as follows:

```
"DX#DY:"
"COMM"
"DX#YDX:"
"COMM"
"DX#XDY:"
"COMM"
  :
"YDY#Y|(XDX+YDY):"
"COMM"
"X|(XDX+YDY)#Y|(XDX+YDY):"
"COMM"
"TERM"
"TERM"
```

Setting up such a problem is quite a job. An accessory program which sets up sets of operators written as:

Q_1
"#"
Q_2
"#"
  :
Q_n
":"

for the sequence $[Q_1,Q_2]$, $[Q_1,Q_3]$, ..., $[Q_{n-1},Q_n]$, equivalent to the above sets of operators and instructions for $n = 8$, is as follows:

```
*  PROGRAM TO SET UP SETS OF OPERATORS FOR COMMUTATION
        &TRIM = 1  ;  &ANCHOR = 1
        OP = TABLE(10)  ;  0 = 1
PEQZE   P = 0  ;  OP<0> = TABLE(5)
INPSQ   OUTPUT = INPUT  :F(END)
        IDENT(OUTPUT,'#')  :F(COLON)
        OP<0><0> = P  ;  0 = 0 + 1  :(PEQZE)
COLON   IDENT(OUTPUT,':')  :S(OPLIM)
        P = P + 1  ;  OP<0><P> = OUTPUT  :(INPSQ)
OPLIM   OP<0><0> = P  ;  OP<0> = 0  ;  0 = 1  ;  Q = 2
PSET1   P = 1
LINOA   OUTPUT = LE(P,OP<0><0>) OP<0><P>  :F(RSET1)
        P = P + 1  :(LINOA)
RSET1   R = 1  ;  OUTPUT = '#'
LINOB   OUTPUT = LE(R,OP<Q><0>) OP<Q><R>  :F(CLOUT)
        R = R + 1  :(LINOB)
CLOUT   OUTPUT = ':'  ;  OUTPUT = 'COMM'
        Q = LT(Q,OP<0>) Q + 1  :S(PSET1)
        0 = LT(0,OP<0> - 1) 0 + 1  :F(END)  ;  Q = 0 + 1  :(PSET1)
END
```

For the inputs:

"DX" (Q_1)

"#"

"DY" (Q_2)

"#"

⋮

"#"

"Y|(XDX+YDY)" (Q_8)

":"

the accessory program produces an output which can be used as a set of inputs in MANDO for determination of all the commutators tabulated above. When using the accessory program, provision must, of course, be made for the output to be put in some accessible file; this can be done by holding the output in a batch run ("run unn hold") on the Stanford system.

The printout for the projective group follows on the next 15 pages:

```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** --*

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  48 MSEC, COUNT:  74 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
DX
#
DY
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (140 MSEC, 227 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  163 MSEC, 289 STATEMENTS.

 NULL RESULT.
```

[1,2]

```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  178 MSEC, COUNT:  311 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
DX
#
YDX
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (278 MSEC, 469 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  299 MSEC, 531 STATEMENTS.

 NULL RESULT.
```

[1,3]

```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  314 MSEC, COUNT:  553 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
DX
#
XDY
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (451 MSEC, 711 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  477 MSEC, 791 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  484 MSEC, 829 STATEMENTS.

 ORDERED RESULT:

 +< 1 >DY


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  497 MSEC, 853 STATEMENTS.
 NULL CONVERSION FOR FUNCTION .
```

```
TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  508 MSEC, 890 STATEMENTS.

SIMPLIFIED RESULT:

+< 1 >DY
```

[1,4]

```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  530 MSEC, COUNT:  919 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
DX
#
XDX
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (632 MSEC, 1075 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  656 MSEC, 1142 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  663 MSEC, 1180 STATEMENTS.

 ORDERED RESULT:

 +< 1 >DX


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  676 MSEC, 1204 STATEMENTS.
 NULL CONVERSION FOR FUNCTION .
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  687 MSEC, 1241 STATEMENTS.

 SIMPLIFIED RESULT:

 +< 1 >DX
```

[1,5]

```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  708 MSEC, COUNT:  1270 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
DX
#
YDY
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (809 MSEC, 1430 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  873 MSEC, 1492 STATEMENTS.

 NULL RESULT.
```

[1,6]

```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  886 MSEC, COUNT:  1514 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
DX
#
X|(XDX+YDY)
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (1042 MSEC, 1753 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  1103 MSEC, 1932 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  1120 MSEC, 2043 STATEMENTS.

 ORDERED RESULT:


 +< Y >DY

 +< 2X >DX


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  1140 MSEC, 2079 STATEMENTS.
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  1160 MSEC, 2139 STATEMENTS.

 SIMPLIFIED RESULT:


 +< Y >DY

 +< 2X >DX


* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  1187 MSEC, COUNT:  2179 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
DX
#
Y|(XDX+YDY)
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (1385 MSEC, 2420 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  1429 MSEC, 2557 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  1435 MSEC, 2603 STATEMENTS.

 ORDERED RESULT:


 +< Y >DX
```


[1,7]

```
TIME AND COUNT AT END OF OUTPUT ROUTINE:  1448 MSEC, 2627 STATEMENTS.
TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  1451 MSEC, 2650 STATEMENTS.

SIMPLIFIED RESULT:

+< Y >DX                                                              [1,8]


* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  1473 MSEC, COUNT:  2679 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
DY
#
YDX
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (1572 MSEC, 2839 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  1598 MSEC, 2917 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  1604 MSEC, 2955 STATEMENTS.

 ORDERED RESULT:

 +< 1 >DX


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  1661 MSEC, 2979 STATEMENTS.
 NULL CONVERSION FOR FUNCTION .
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  1671 MSEC, 3016 STATEMENTS.

 SIMPLIFIED RESULT:

 +< 1 >DX                                                              [2,3]


* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  1692 MSEC, COUNT:  3045 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
DY
#
XDY
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (1790 MSEC, 3205 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  1812 MSEC, 3267 STATEMENTS.
```

```
NULL RESULT.                                                    [2,4]

* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  1826 MSEC, COUNT:  3289 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
DY
#
XDX
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (1928 MSEC, 3447 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  1952 MSEC, 3509 STATEMENTS.

 NULL RESULT.
                                                                [2,5]

* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  1967 MSEC, COUNT:  3531 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
DY
#
YDY
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (2110 MSEC, 3693 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  2134 MSEC, 3760 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  2140 MSEC, 3798 STATEMENTS.

 ORDERED RESULT:

 +< 1 >DY


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  2152 MSEC, 3822 STATEMENTS.
 NULL CONVERSION FOR FUNCTION .
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  2162 MSEC, 3859 STATEMENTS.

 SIMPLIFIED RESULT:

 +< 1 >DY                                                       [2,6]


* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  2184 MSEC, COUNT:  3888 STATEMENTS *$$*
```

```
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
DY
#
X|(XDX+YDY)
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (2337 MSEC, 4129 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  2378 MSEC, 4264 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  2384 MSEC, 4310 STATEMENTS.

 ORDERED RESULT:


 +< X >DY


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  2397 MSEC, 4334 STATEMENTS.
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  2400 MSEC, 4357 STATEMENTS.

 SIMPLIFIED RESULT:

 +< X >DY                                                                    [2,7]



* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  2468 MSEC, COUNT:  4386 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
DY
#
Y|(XDX+YDY)
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (2627 MSEC, 4629 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  2685 MSEC, 4804 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  2701 MSEC, 4915 STATEMENTS.

 ORDERED RESULT:


 +< 2Y >DY

 +< X >DX


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  2720 MSEC, 4951 STATEMENTS.
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  2724 MSEC, 4984 STATEMENTS.

 SIMPLIFIED RESULT:                                                          [2,8]


 +< 2Y >DY
```

```
+< X >DX



*$$* BEGINNING OF INPUT SEQUENCE, TIME:  2750 MSEC, COUNT:  5024 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
YDX
#
XDY
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (2900 MSEC, 5191 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  2928 MSEC, 5264 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  2943 MSEC, 5361 STATEMENTS.

 ORDERED RESULT:


 +< Y >DY

 +< -X >DX


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  2961 MSEC, 5397 STATEMENTS.
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  2966 MSEC, 5430 STATEMENTS.

 SIMPLIFIED RESULT:

 +< Y >DY                                                    [3,4]

 +< -X >DX



*$$* BEGINNING OF INPUT SEQUENCE, TIME:  2992 MSEC, COUNT:  5470 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
YDX
#
XDX
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (3096 MSEC, 5635 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  3119 MSEC, 5701 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  3125 MSEC, 5739 STATEMENTS.

 ORDERED RESULT:
```

```
+< Y >DX


TIME AND COUNT AT END OF OUTPUT ROUTINE:  3140 MSEC, 5763 STATEMENTS.
TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  3144 MSEC, 5784 STATEMENTS.

SIMPLIFIED RESULT:

+< Y >DX                                                              [3,5]


* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  3164 MSEC, COUNT:  5813 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
YDX
#
YDY
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (3310 MSEC, 5982 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  3335 MSEC, 6051 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  3341 MSEC, 6087 STATEMENTS.

 ORDERED RESULT:

 +< -Y >DX


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  3354 MSEC, 6111 STATEMENTS.
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  3357 MSEC, 6132 STATEMENTS.

 SIMPLIFIED RESULT:

 +< -Y >DX                                                            [3,6]


* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  3378 MSEC, COUNT:  6161 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
YDX
#
X|(XDX+YDY)
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (3534 MSEC, 6409 STATEMENTS)
```

```
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  3572 MSEC, 6536 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  3592 MSEC, 6667 STATEMENTS.

 ORDERED RESULT:


 +< YY >DY

 +< 2YX -XY >DX


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  3615 MSEC, 6715 STATEMENTS.
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  3695 MSEC, 6809 STATEMENTS.

 SIMPLIFIED RESULT:


 +< Y@(2) >DY                                                    [3,7]

 +< YX >DX



* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  3721 MSEC, COUNT:  6849 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
YDX
#
Y|(XDX+YDY)
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (3877 MSEC, 7099 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  3913 MSEC, 7221 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  3925 MSEC, 7300 STATEMENTS.

 ORDERED RESULT:


 +< YY -YY >DX


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  3943 MSEC, 7336 STATEMENTS.

 NULL RESULT.                                                    [3,8]


* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  3961 MSEC, COUNT:  7381 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
XDY
```

```
#
XDX
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (4106 MSEC, 7546 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  4130 MSEC, 7615 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  4136 MSEC, 7651 STATEMENTS.

 ORDERED RESULT:


 +< -X >DY


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  4148 MSEC, 7675 STATEMENTS.
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  4152 MSEC, 7696 STATEMENTS.

 SIMPLIFIED RESULT:


 +< -X >DY
```

[4,5]

```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  4173 MSEC, COUNT:  7725 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
XDY
#
YDY
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (4275 MSEC, 7894 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  4298 MSEC, 7960 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  4304 MSEC, 7998 STATEMENTS.

 ORDERED RESULT:


 +< X >DY


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  4317 MSEC, 8022 STATEMENTS.
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  4320 MSEC, 8043 STATEMENTS.

 SIMPLIFIED RESULT:


 +< X >DY
```

[4,6]

```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  4340 MSEC, COUNT:  8072 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
XDY
#
X|(XDX+YDY)
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (4544 MSEC, 8320 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  4580 MSEC, 8442 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  4591 MSEC, 8523 STATEMENTS.

 ORDERED RESULT:


 +< -XX +XX >DY


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  4607 MSEC, 8559 STATEMENTS.

 NULL RESULT.
```

[4,7]


```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  4638 MSEC, COUNT:  8623 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
XDY
#
Y|(XDX+YDY)
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (4844 MSEC, 8873 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  4882 MSEC, 9000 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  4902 MSEC, 9133 STATEMENTS.

 ORDERED RESULT:


 +< -YX +2XY >DY

 +< XX >DX


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  4925 MSEC, 9181 STATEMENTS.
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  4935 MSEC, 9231 STATEMENTS.

 SIMPLIFIED RESULT:


 +< YX >DY
```

[4,8]

```
+< X@(2) >DX
```

```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  4961 MSEC, COUNT:  9271 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
XDX
#
YDY
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (5062 MSEC, 9438 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  5085 MSEC, 9500 STATEMENTS.

 NULL RESULT.
```

[5,6]

```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  5098 MSEC, COUNT:  9522 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
XDX
#
X|(XDX+YDY)
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (5299 MSEC, 9768 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  5338 MSEC, 9893 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  5359 MSEC, 10027 STATEMENTS.

 ORDERED RESULT:

 +< XY >DY

 +< 2XX -XX >DX


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  5381 MSEC, 10075 STATEMENTS.
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  5391 MSEC, 10121 STATEMENTS.

 SIMPLIFIED RESULT:

 +< YX >DY
```

[5,7]

```
 +< X@(2) >DX
```

```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  5420 MSEC, COUNT:  10161 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
XDX
#
Y|(XDX+YDY)
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (5625 MSEC, 10409 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  5664 MSEC, 10530 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  5675 MSEC, 10611 STATEMENTS.

 ORDERED RESULT:


 +< XY -YX >DX


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  5692 MSEC, 10647 STATEMENTS.

 NULL RESULT.
```

[5,8]

```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  5712 MSEC, COUNT:  10690 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
YDY
#
X|(XDX+YDY)
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (5872 MSEC, 10940 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  5908 MSEC, 11061 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  5919 MSEC, 11140 STATEMENTS.

 ORDERED RESULT:


 +< YX -XY >DY


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  5935 MSEC, 11176 STATEMENTS.

 NULL RESULT.
```

[6,7]

```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  5952 MSEC, COUNT:  11219 STATEMENTS *$$*
```

```
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
YDY
#
Y|(XDX+YDY)
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (6156 MSEC, 11471 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  6194 MSEC, 11596 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  6214 MSEC, 11727 STATEMENTS.

 ORDERED RESULT:


 +< 2YY -YY >DY

 +< YX >DX


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  6236 MSEC, 11775 STATEMENTS.
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  6246 MSEC, 11821 STATEMENTS.

 SIMPLIFIED RESULT:


 +< Y@(2) >DY

 +< YX >DX
```

[6,8]

-36-

```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  6274 MSEC, COUNT:  11861 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
X|(XDX+YDY)
#
Y|(XDX+YDY)
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (6535 MSEC, 12192 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  6595 MSEC, 12427 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  6628 MSEC, 12641 STATEMENTS.

 ORDERED RESULT:


 +< -YXY +2XYY -YYX >DY

 +< XXY -2YXX +XYX >DX


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  6662 MSEC, 12717 STATEMENTS.

 NULL RESULT.
```

[7,8]

```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  6806 MSEC, COUNT:  12967 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
TERM
```

The compilation and termination statistics for this run are as follows:

COMPILATION TIME	4.628 SECONDS	NORMAL TERMINATION IN STATEMENT	1040
MEMORY USAGE (DECIMAL BYTES)		NUMBER OF STATEMENTS EXECUTED	12972
CODE:	92538	EXECUTION TIME (SECONDS)	6.816
STRINGS:	4831	MSEC/STATEMENT	0.525
VARIABLES:	14880	NUMBER OF STORAGE REGENERATIONS	17
CONSTANTS:	3416		
TOTAL:	115665		
AVAILABLE:	91599		

Normally one would not go to all the trouble to verify such a simple set of commutators by this procedure, but the result clearly shows the format in which the answers to more complicated, though perhaps more restricted, problems is to be found, in a context where the only difficulty is the language in which the results are cast. After the program has run, making remarks on the printout helps to identify the separate results for later reference.

Example 2. The seven second-extended generators derived by the Lie method* from the differential equation $U''' = 0$** are: $Q_1 = x^2\partial_u + 2x\partial_{u'} + 2\partial_{u''}$, $Q_2 = x^2\partial_x + 2xu\partial_u + 2u\partial_{u'} + (2u' - 2xu'')\partial_{u''}$, $Q_3 = x\partial_u + \partial_{u'}$, $Q_4 = x\partial_x - u'\partial_{u'} - 2u''\partial_{u''}$, $Q_5 = \partial_x$, $Q_6 = u\partial_u + u'\partial_{u'} + u''\partial_{u''}$, and $Q_7 = \partial_u$. It is expected that these operators form a closed Lie algebra. That this is in fact the case is substantiated by the run on the following 9 pages with the compilation and termination statistics:

COMPILATION TIME	4.570 SECONDS	NORMAL TERMINATION IN STATEMENT	1016
MEMORY USAGE (DECIMAL BYTES)		NUMBER OF STATEMENTS EXECUTED	16834
CODE:	93394	EXECUTION TIME (SECONDS)	11.466
STRINGS:	4766	MSEC/STATEMENT	0.681
VARIABLES:	14400	NUMBER OF STORAGE REGENERATIONS	53
CONSTANTS:	3792		
TOTAL:	116352		
AVAILABLE:	94190		

* The second extension is explained in pp. 83 to 85, Cohen (2).

** The derivation is compliments of Suzanne Davison, who studies the equation $U''' = 0$ in more detail in her M.Sc. thesis, University of the Pacific (1973).

```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  48 MSEC, COUNT:  69 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
X@(2)DU+2XDV+2DW
#
X@(2)DX+2XUDU+2UDV+(2V-2XW)DW
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (663 MSEC, 539 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  839 MSEC, 1373 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  885 MSEC, 1690 STATEMENTS.

 ORDERED RESULT:


 +< 4X -4X >DW

 +< -2X@(2) +2X@(2) >DV

 +< -2X@(2)X +2X@(2)X >DU


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  920 MSEC, 1774 STATEMENTS.

 NULL RESULT.
```

[1,2]


```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  1064 MSEC, COUNT:  1903 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
X@(2)DU+2XDV+2DW
#
XDU+DV
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (1412 MSEC, 2196 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  1474 MSEC, 2502 STATEMENTS.

 NULL RESULT.
```

[1,3]

```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  1489 MSEC, COUNT:  2544 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
X@(2)DU+2XDV+2DW
#
XDX-VDV-2WDW
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
```

```
 (TIME,COUNT):  (1912 MSEC, 2698 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  2045 MSEC, 3398 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  2076 MSEC, 3639 STATEMENTS.

 ORDERED RESULT:


 +< -4 >DW

 +< -2X -2X >DV

 +< -2XX >DU


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  2105 MSEC, 3699 STATEMENTS.
 NULL CONVERSION FOR FUNCTION .
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  2253 MSEC, 3799 STATEMENTS.

 SIMPLIFIED RESULT:


 +< -4 >DW

 +< -4X >DV

 +< -2X@(2) >DU


* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  2285 MSEC, COUNT:  3850 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
X@(2)DU+2XDV+2DW
#
DX
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (2547 MSEC, 4097 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  2594 MSEC, 4273 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  2610 MSEC, 4392 STATEMENTS.

 ORDERED RESULT:


 +< -2 >DV

 +< -2X >DU


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  2629 MSEC, 4428 STATEMENTS.
 NULL CONVERSION FOR FUNCTION .
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  2737 MSEC, 4490 STATEMENTS.
```

[1,4]

-04-

```
SIMPLIFIED RESULT:

+< -2 >DV

+< -2X >DU
```

[1,5]

```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  2764 MSEC, COUNT:  4530 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
X@(2)DU+2XDV+2DW
#
UDU+VDV+WDW
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (3184 MSEC, 4888 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  3317 MSEC, 5384 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  3343 MSEC, 5599 STATEMENTS.

 ORDERED RESULT:

 +< 2 >DW

 +< 2X >DV

 +< X@(2) >DU


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  3368 MSEC, 5647 STATEMENTS.
 NULL CONVERSION FOR FUNCTION .
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  3456 MSEC, 5735 STATEMENTS.

 SIMPLIFIED RESULT:

 +< 2 >DW

 +< 2X >DV

 +< X@(2) >DU
```

[1,6]

```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  3489 MSEC, COUNT:  5786 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
X@(2)DU+2XDV+2DW
#
DU
```

```
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (3791 MSEC, 6035 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  3833 MSEC, 6199 STATEMENTS.

 NULL RESULT.
```

[1,7]

```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  3847 MSEC, COUNT:  6229 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
X@(2)DX+2XUDU+2UDV+(2V-2XW)DW
#
XDU+DV
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (4415 MSEC, 6649 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  4511 MSEC, 7135 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  4542 MSEC, 7367 STATEMENTS.

 ORDERED RESULT:


 +< -2 >DW

 +< -2X >DV

 +< X@(2) -2XX >DU


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  4570 MSEC, 7427 STATEMENTS.
 NULL CONVERSION FOR FUNCTION .
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  4783 MSEC, 7542 STATEMENTS.

 SIMPLIFIED RESULT:


 +< -2 >DW

 +< -2X >DV

 +< -X@(2) >DU
```

[2,3]

```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  4816 MSEC, COUNT:  7593 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
X@(2)DX+2XUDU+2UDV+(2V-2XW)DW
#
XDX-VDV-2WDW
```

```
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (5463 MSEC, 8074 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  5664 MSEC, 8815 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  5723 MSEC, 9227 STATEMENTS.

 ORDERED RESULT:


 +< 2XW +2V -4V +4XW -4WX >DW

 +< -2U >DV

 +< -2XU >DU

 +< X@(2) -2XX >DX


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  5772 MSEC, 9335 STATEMENTS.
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  6131 MSEC, 9579 STATEMENTS.

 SIMPLIFIED RESULT:


 +< 2XW -2V >DW

 +< -2U >DV

 +< -2XU >DU

 +< -X@(2) >DX



* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  6173 MSEC, COUNT:  9654 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
X@(2)DX+2XUDU+2UDV+(2V-2XW)DW
#
DX
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (6703 MSEC, 10028 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  6764 MSEC, 10270 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  6789 MSEC, 10443 STATEMENTS.

 ORDERED RESULT:


 +< 2W >DW

 +< -2U >DU

 +< -2X >DX
```

[2,4]

```
TIME AND COUNT AT END OF OUTPUT ROUTINE:  6814 MSEC, 10491 STATEMENTS.
TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  6965 MSEC, 10578 STATEMENTS.

SIMPLIFIED RESULT:


+< 2W >DW

+< -2U >DU

+< -2X >DX
```

[2,5]

```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  6998 MSEC, COUNT:  10629 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
X@(2)DX+2XUDU+2UDV+(2V-2XW)DW
#
UDU+VDV+WDW
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (7684 MSEC, 11114 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  7809 MSEC, 11798 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  7856 MSEC, 12142 STATEMENTS.

 ORDERED RESULT:


 +< -2V +2V -2XW +2WX >DW

 +< -2U +2U >DV

 +< -2UX +2XU >DU


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  7900 MSEC, 12242 STATEMENTS.

 NULL RESULT.
```

[2,6]

```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  8194 MSEC, COUNT:  12449 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
X@(2)DX+2XUDU+2UDV+(2V-2XW)DW
#
DU
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
```

```
 (TIME,COUNT):  (8720 MSEC, 12825 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  8787 MSEC, 13085 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  8806 MSEC, 13212 STATEMENTS.

 ORDERED RESULT:


 +< -2 >DV

 +< -2X >DU


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  8826 MSEC, 13248 STATEMENTS.
 NULL CONVERSION FOR FUNCTION .
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  8948 MSEC, 13310 STATEMENTS.

 SIMPLIFIED RESULT:


 +< -2 >DV

 +< -2X >DU
```

[2,7]


```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  8977 MSEC, COUNT:  13350 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
XDU+DV
#
XDX-VDV-2WDW
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (9386 MSEC, 13658 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  9458 MSEC, 13974 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  9476 MSEC, 14117 STATEMENTS.

 ORDERED RESULT:


 +< -1 >DV

 +< -X >DU


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  9496 MSEC, 14153 STATEMENTS.
 NULL CONVERSION FOR FUNCTION .
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  9609 MSEC, 14215 STATEMENTS.

 SIMPLIFIED RESULT:


 +< -1 >DV
```

[3,4]

```
 +< -X >DU



* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  9635 MSEC, COUNT:  14255 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
XDU+DV
#
DX
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (9850 MSEC, 14456 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  9884 MSEC, 14576 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  9890 MSEC, 14622 STATEMENTS.

 ORDERED RESULT:


 +< -1 >DU


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  9903 MSEC, 14646 STATEMENTS.
 NULL CONVERSION FOR FUNCTION .
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  9948 MSEC, 14686 STATEMENTS.

 SIMPLIFIED RESULT:


 +< -1 >DU                                                    [3,5]



* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  9968 MSEC, COUNT:  14715 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
XDU+DV
#
UDU+VDV+WDW
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (10359 MSEC, 15027 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  10486 MSEC, 15339 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  10506 MSEC, 15482 STATEMENTS.

 ORDERED RESULT:


 +< 1 >DV
```

```
+< X >DU


TIME AND COUNT AT END OF OUTPUT ROUTINE:  10530 MSEC, 15518 STATEMENTS.
NULL CONVERSION FOR FUNCTION .
TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  10605 MSEC, 15580 STATEMENTS.

SIMPLIFIED RESULT:


+< 1 >DV

+< X >DU                                                              [3,6]


* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  10631 MSEC, COUNT:  15620 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
XDX-VDV-2WDW
#
UDU+VDV+WDW
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (11144 MSEC, 15993 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  11235 MSEC, 16459 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  11265 MSEC, 16669 STATEMENTS.

 ORDERED RESULT:


 +< -2W +2W >DW

 +< -V +V >DV


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  11293 MSEC, 16729 STATEMENTS.

 NULL RESULT.                                                         [4,6]


* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  11457 MSEC, COUNT:  16830 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
TERM
```

On the printouts the inputs are the lines which do not begin with an asterisk but which are left justified, as are the lines beginning with asterisks:

```
"X@(2)DU+2XDV+2DW"
"#"
"X@(2)DX+2XUDU+2UDV+(2V-2XW)DW"
":"
"COMM"
"X@(2)DU+2XDV+2DW"
"#"
"XDU+DV"
":"
"COMM"
  ⋮
```

(the substitutions V = u' and W = u" were made). The commutators may be tabulated as follows:

$[\downarrow,\rightarrow]$	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6
Q_2	0	-				
Q_3	0	Q_1	-			
Q_4	$2Q_1$	Q_2	Q_3	-		
Q_5	$2Q_3$	$2(Q_4+Q_6)$	Q_7	Q_5	-	
Q_6	$-Q_1$	0	$-Q_3$	0	0	-
Q_7	0	$2Q_3$	0	0	0	Q_7

Commutators [5,4], [6,5], [7,3], [7,4], [7,5] and [7,6] were not included in the run because they were thought to be zero. Later examination revealed [5,4] and [7,6] to be nonzero, however. Note that the calculated results are the transposes (negatives for commutators) of those tabulated.

Example 3. To investigate the finite tarnsformation of a point x corresponding to the dilatation operator $ax\partial_x$, the operator $\exp(ax\partial_x)$ is expanded to order 8 and applied, term by term, to function x. The set of inputs is as follows:

"AXDX#AXDX:"	Q, Q
"SAVE"	
"APLY X"	Qx
"PROD"	Q^2
"APLY X"	Q^2x
"PROD"	Q^3
"APLY X"	Q^3x
⋮	
"PROD"	Q^8
"APLY X"	Q^8x
"OPER"	
"TERM"	
"TERM"	

The run on the next 10 pages had compilation and termination statistics as follows:

```
COMPILATION TIME  4.481  SECONDS
MEMORY USAGE (DECIMAL BYTES)
CODE:              93682
STRINGS:            4769
VARIABLES:         14464
CONSTANTS:          3800
TOTAL:            116715
AVAILABLE:         93897

NORMAL TERMINATION IN STATEMENT     1020
NUMBER OF STATEMENTS EXECUTED      10076
EXECUTION TIME (SECONDS)            5.228
MSEC/STATEMENT                      0.518
NUMBER OF STORAGE REGENERATIONS        20
```

```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  50 MSEC, COUNT:  71 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
AXDX#AXCX:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (184 MSEC, 201 STATEMENTS)
SAVE

* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* ENTRY INTO EXECUTION CYCLE *$$*
 "COMM", "ANTI", "PROD" OR "APLY":
APLY X
 TIME AND COUNT AT END OF APPLICATION ROUTINE AND BEGINNING OF SIMPLIFICATION (SIMCO):  287 MSEC, 290 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  293 MSEC, 326 STATEMENTS.

 ORDERED RESULT:


 +< AX >


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  305 MSEC, 346 STATEMENTS.
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  338 MSEC, 390 STATEMENTS.

 SIMPLIFIED RESULT:


 +< AX >


* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* CONTINUATION OF EXECUTION CYCLE, TIME:  354 MSEC, COUNT:  412 STATEMENTS *$$*
 "COMM", "ANTI", "PROD", "APLY" OR "OPER":
PROD
 TIME AND COUNT AT END OF PRODUCT ROUTINE AND BEGINNING OF SIMPLIFICATION (SIMCO):  375 MSEC, 479 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  384 MSEC, 531 STATEMENTS.

 ORDERED RESULT:


 +< AAXX >CXCX


 +< AAX >CX


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  405 MSEC, 572 STATEMENTS.
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  457 MSEC, 649 STATEMENTS.

 SIMPLIFIED RESULT:
```

Qx

```
+< A@(2)X@(2) >DXDX

+< A@(2)X >DX
```

Q^2

```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* CONTINUATION OF EXECUTION CYCLE, TIME:  483 MSEC, COUNT:  693 STATEMENTS *$$*
 "COMM", "ANTI", "PROD", "APLY" OR "OPER":
APLY X
 TIME AND COUNT AT END OF APPLICATION ROUTINE AND BEGINNING OF SIMPLIFICATION (SIMCO):  596 MSEC, 822 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  602 MSEC, 862 STATEMENTS.

 ORDERED RESULT:

 +< A@(2)X >


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  614 MSEC, 882 STATEMENTS.
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  620 MSEC, 907 STATEMENTS.

 SIMPLIFIED RESULT:

 +< A@(2)X >
```

$Q^2 X$


```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* CONTINUATION OF EXECUTION CYCLE, TIME:  637 MSEC, COUNT:  929 STATEMENTS *$$*
 "COMM", "ANTI", "PROD", "APLY" OR "OPER":
PROD
 TIME AND COUNT AT END OF PRODUCT ROUTINE AND BEGINNING OF SIMPLIFICATION (SIMCO):  683 MSEC, 1056 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  700 MSEC, 1167 STATEMENTS.

 ORDERED RESULT:

 +< AA@(2)XX@(2) >DXDXDX

 +< 2AA@(2)XX +AA@(2)XX >DXDX

 +< AA@(2)X >DX


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  732 MSEC, 1239 STATEMENTS.
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  835 MSEC, 1358 STATEMENTS.

 SIMPLIFIED RESULT:
```

```
+< A@(3)X@(3) >DXDXDX

+< 3A@(3)X@(2) >DXDX

+< A@(3)X >DX
```

Q^3

```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* CONTINUATION OF EXECUTION CYCLE, TIME:  866 MSEC, COUNT:  1420 STATEMENTS *$$*
 "COMM", "ANTI", "PROD", "APLY" OR "OPER":
APLY X
 TIME AND COUNT AT END OF APPLICATION ROUTINE AND BEGINNING OF SIMPLIFICATION (SIMCO):  953 MSEC, 1592 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  959 MSEC, 1636 STATEMENTS.

 ORDERED RESULT:

 +< A@(3)X >


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  971 MSEC, 1656 STATEMENTS.
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  976 MSEC, 1681 STATEMENTS.

 SIMPLIFIED RESULT:
```

Q^3X

```
 +< A@(3)X >


* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* CONTINUATION OF EXECUTION CYCLE, TIME:  996 MSEC, COUNT:  1703 STATEMENTS *$$*
 "COMM", "ANTI", "PROD", "APLY" OR "OPER":
PROD
 TIME AND COUNT AT END OF PRODUCT ROUTINE AND BEGINNING OF SIMPLIFICATION (SIMCO):  1067 MSEC, 1893 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  1092 MSEC, 2063 STATEMENTS.

 ORDERED RESULT:

 +< AA@(3)XX@(3) >DXDXDXDX

 +< 3AA@(3)XX@(2) +3AA@(3)XX@(2) >DXDXDX

 +< 6AA@(3)XX +AA@(3)XX >DXDX

 +< AA@(3)X >DX
```

```
TIME AND COUNT AT END OF OUTPUT ROUTINE:  1136 MSEC, 2168 STATEMENTS.
TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  1255 MSEC, 2327 STATEMENTS.

SIMPLIFIED RESULT:

+< A@(4)X@(4) >DXDXDXDX

+< 6@@(4)X@(3) >DXDXDX

+< 7@@(4)X@(2) >DXDX

+< A@(4)X >DX

* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* CONTINUATION OF EXECUTION CYCLE, TIME:  1294 MSEC, COUNT:  2409 STATEMENTS *$$*
 "COMM", "ANTI", "PROD", "APLY" OR "OPER":
APLY X
 TIME AND COUNT AT END OF APPLICATION ROUTINE AND BEGINNING OF SIMPLIFICATION (SIMCO):  1435 MSEC, 2629 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  1442 MSEC, 2677 STATEMENTS.

 ORDERED RESULT:

 +< A@(4)X >


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  1454 MSEC, 2697 STATEMENTS.
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  1459 MSEC, 2722 STATEMENTS.

 SIMPLIFIED RESULT:

 +< A@(4)X >

* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* CONTINUATION OF EXECUTION CYCLE, TIME:  1476 MSEC, COUNT:  2744 STATEMENTS *$$*
 "COMM", "ANTI", "PROD", "APLY" OR "OPER":
PROD
 TIME AND COUNT AT END OF PRODUCT ROUTINE AND BEGINNING OF SIMPLIFICATION (SIMCO):  1572 MSEC, 2997 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  1604 MSEC, 3226 STATEMENTS.

 ORDERED RESULT:

 +< AA@(4)XX@(4) >DXDXDXDXDX
```

Q^4

$Q^4{}_X$

```
+< 4AA@(4)XX@(3) +6AA@(4)XX@(3) >DXDXDXDX

+< 18AA@(4)XX@(2) +7AA@(4)XX@(2) >DXDXDX

+< 14AA@(4)XX +AA@(4)XX >DXDX

+< AA@(4)X >DX


TIME AND COUNT AT END OF OUTPUT ROUTINE:  1660 MSEC, 3366 STATEMENTS.
TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  1794 MSEC, 3565 STATEMENTS.

SIMPLIFIED RESULT:

+< A@(5)X@(5) >DXDXDXDXDX

+< 10A@(5)X@(4) >DXDXDXDX

+< 25A@(5)X@(3) >DXDXDX

+< 15A@(5)X@(2) >DXDX

+< A@(5)X >DX


* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* CONTINUATION OF EXECUTION CYCLE, TIME:  1842 MSEC, COUNT:  3669 STATEMENTS *$$*
 "COMM", "ANTI", "PROD", "APLY" OR "OPER":
APLY X
 TIME AND COUNT AT END OF APPLICATION ROUTINE AND BEGINNING OF SIMPLIFICATION (SIMCO):  1953 MSEC, 3939 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  1959 MSEC, 3991 STATEMENTS.

 ORDERED RESULT:

 +< A@(5)X >


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  1972 MSEC, 4011 STATEMENTS.
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  1986 MSEC, 4047 STATEMENTS.

 SIMPLIFIED RESULT:

 +< A@(5)X >
```

Q^5

Q^5_X

```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$** CONTINUATION OF EXECUTION CYCLE, TIME:  2003 MSEC, COUNT:  4069 STATEMENTS *$$*
 "COMM", "ANTI", "PROD", "APLY" OR "OPER":
PROD
 TIME AND COUNT AT END OF PRODUCT ROUTINE AND BEGINNING OF SIMPLIFICATION (SIMCO):  2168 MSEC, 4385 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  2209 MSEC, 4673 STATEMENTS.

 ORDERED RESULT:

 +< AA@(5)XX@(5) >DXDXDXDXDXDX

 +< 5AA@(5)XX@(4) +10AA@(5)XX@(4) >DXDXDXDXDX

 +< 40AA@(5)XX@(3) +25AA@(5)XX@(3) >DXDXDXDX

 +< 75AA@(5)XX@(2) +15AA@(5)XX@(2) >DXDXDX

 +< 30AA@(5)XX +AA@(5)XX >DXDX

 +< AA@(5)X >DX


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  2278 MSEC, 4850 STATEMENTS.
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  2620 MSEC, 5161 STATEMENTS.

 SIMPLIFIED RESULT:

 +< A@(6)X@(6) >DXDXDXDXDXDX

 +< 15A@(6)X@(5) >DXDXDXDXDX

 +< 65A@(6)X@(4) >DXDXDXDX

 +< 90A@(6)X@(3) >DXDXDX

 +< 31A@(6)X@(2) >DXDX

 +< A@(6)X >DX
```

Q6

```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* CONTINUATION OF EXECUTION CYCLE, TIME:  2675 MSEC, COUNT:  5289 STATEMENTS *$$*
 "COMM", "ANTI", "PROD", "APLY" OR "OPER":
APLY X
 TIME AND COUNT AT END OF APPLICATION ROUTINE AND BEGINNING OF SIMPLIFICATION (SIMCO):  2840 MSEC, 5611 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  2846 MSEC, 5667 STATEMENTS.

 ORDERED RESULT:


 +< A@(6)X >


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  2859 MSEC, 5667 STATEMENTS.
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  2893 MSEC, 5733 STATEMENTS.

 SIMPLIFIED RESULT:

                                                                                  Q^6 X

 +< A@(6)X >



* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* CONTINUATION OF EXECUTION CYCLE, TIME:  2910 MSEC, COUNT:  5755 STATEMENTS *$$*
 "COMM", "ANTI", "PROD", "APLY" OR "OPER":
PROD
 TIME AND COUNT AT END OF PRODUCT ROUTINE AND BEGINNING OF SIMPLIFICATION (SIMCO):  3105 MSEC, 6134 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  3154 MSEC, 6481 STATEMENTS.

 ORDERED RESULT:


 +< AA@(6)XX@(6) >DXDXDXDXDXDXDX


 +< 6AA@(6)XX@(5) +15AA@(6)XX@(5) >DXDXDXDXDXDX


 +< 75AA@(6)XX@(4) +65AA@(6)XX@(4) >DXDXDXDXDX


 +< 260AA@(6)XX@(3) +90AA@(6)XX@(3) >DXDXDXDX


 +< 270AA@(6)XX@(2) +31AA@(6)XX@(2) >DXDXDX


 +< 62AA@(6)XX +AA@(6)XX >DXDX


 +< AA@(6)X >DX


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  3235 MSEC, 6697 STATEMENTS.
```

TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT: 3688 MSEC, 7064 STATEMENTS.

SIMPLIFIED RESULT:

+< A@(7)X@(7) >DXDXDXDXDXDXDX

+< 21A@(7)X@(6) >DXDXDXDXDXDX

+< 140A@(7)X@(5) >DXDXDXDXDX

+< 350A@(7)X@(4) >DXDXDXDX

+< 301A@(7)X@(3) >DXDXDX

+< 63A@(7)X@(2) >DXDX

+< A@(7)X >DX

Q^7

* -- *

$$* CONTINUATION OF EXECUTION CYCLE, TIME: 3752 MSEC, COUNT: 7218 STATEMENTS *$$
"COMM", "ANTI", "PROD", "APLY" OR "OPER":
APLY X
TIME AND COUNT AT END OF APPLICATION ROUTINE AND BEGINNING OF SIMPLIFICATION (SIMCO): 3887 MSEC, 7594 STATEMENTS.
TIME AND COUNT AT END OF ORDERING OF RESULT: 3894 MSEC, 7654 STATEMENTS.

ORDERED RESULT:

+< A@(7)X >

TIME AND COUNT AT END OF OUTPUT ROUTINE: 3906 MSEC, 7674 STATEMENTS.
TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT: 3990 MSEC, 7720 STATEMENTS.

SIMPLIFIED RESULT:

+< A@(7)X >

Q^7_X

* -- *

$$* CONTINUATION OF EXECUTION CYCLE, TIME: 4007 MSEC, COUNT: 7742 STATEMENTS *$$
"COMM", "ANTI", "PROD", "APLY" OR "OPER":
PROD
TIME AND COUNT AT END OF PRODUCT ROUTINE AND BEGINNING OF SIMPLIFICATION (SIMCO): 4242 MSEC, 8210 STATEMENTS.

```
TIME AND COUNT AT END OF ORDERING OF RESULT:  4299 MSEC, 8616 STATEMENTS.
ORDERED RESULT:

+< AA@(7)XX@(7) >DXDXDXDXDXDXDXDX

+< 7AA@(7)XX@(6) +21AA@(7)XX@(6) >DXDXDXDXDXDXDX

+< 126AA@(7)XX@(5) +140AA@(7)XX@(5) >DXDXDXDXDXDX

+< 700AA@(7)XX@(4) +350AA@(7)XX@(4) >DXDXDXDXDX

+< 1400AA@(7)XX@(3) +301AA@(7)XX@(3) >DXDXDXDX

+< 903AA@(7)XX@(2) +63AA@(7)XX@(2) >DXDXDX

+< 126AA@(7)XX +AA@(7)XX >DXDX

+< AA@(7)X >DX


TIME AND COUNT AT END OF OUTPUT ROUTINE:  4390 MSEC, 8873 STATEMENTS.
TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  4875 MSEC, 9296 STATEMENTS.
SIMPLIFIED RESULT:

+< A@(8)X@(9) >DXDXDXDXDXDXDXDX

+< 28A@(8)X@(7) >DXDXDXDXDXDXDX

+< 266A@(8)X@(6) >DXDXDXDXDXDX

+< 1050A@(8)X@(5) >DXDXDXDXDX

+< 1701A@(8)X@(4) >DXDXDXDX

+< 966A@(8)X@(3) >DXDXDX

+< 127A@(8)X@(2) >DXDX

+< A@(8)X >DX
```

```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* CONTINUATION OF EXECUTION CYCLE, TIME:  4946 MSEC, COUNT:  9478 STATEMENTS *$$*
 "COMM", "ANTI", "PROD", "APLY" OR "OPER":
APLY X
 TIME AND COUNT AT END OF APPLICATION ROUTINE AND BEGINNING OF SIMPLIFICATION (SIMCO):  5135 MSEC, 9910 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  5142 MSEC, 9974 STATEMENTS.

 ORDERED RESULT:


 +< A@(8)X >


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  5155 MSEC, 9994 STATEMENTS.
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  5191 MSEC, 10040 STATEMENTS.

 SIMPLIFIED RESULT:


 +< A@(8)X >

* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* CONTINUATION OF EXECUTION CYCLE, TIME:  5208 MSEC, COUNT:  10062 STATEMENTS *$$*
 "COMM", "ANTI", "PROD", "APLY" OR "OPER":
OPER

* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  5220 MSEC, COUNT:  10072 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
TERM
```

Q^{8}_{X}

In the actual expansion, the n-th order term is divided by n!; this should be kept in mind when interpreting the result. In this case it is evident that $e^{Q}x = e^{a}x$. This example illustrates how quickly the program can evaluate some fairly complicated higher-order operators; Q^8 was evaluated in 4.9 seconds. The current version of the program would be even faster (maybe a second less).

Example 4. To test the upper limit of the program's capacity, the commutation of two very large second-order operators is performed. The operators selected are ones which have been identified with elements of the O(4,2) algebra* of the dynamical group of the hydrogenlike atom. The commutator is thus expected to be expressed simply in terms of other operators of the set. The two operators selected are:

$$Q_{11} = 2ie^{-it+iv}\{xy\partial_r\partial_x + y^3/r\partial_x\partial_x - i/y\partial_r\partial_v + 1/ry\partial_v\partial_v + (1 - \tfrac{1}{2}r)y\partial_r - \tfrac{1}{2}(4/r - 1)xy\partial_x - i/2y\partial_v - \tfrac{1}{2}iy\partial_t - \tfrac{1}{4}ry\}$$

and

$$Q_{14} = 2ie^{it}\{-y^2\partial_r\partial_x + xy^2/r\partial_x\partial_x + x/ry^2\partial_v\partial_v + (1 + \tfrac{1}{2}r)x\partial_r + \tfrac{1}{2}(y^2 - 4x^2/r)\partial_x - \tfrac{1}{2}ix\partial_t - \tfrac{1}{4}rx\}$$

(That the coefficient of ∂_t in Q_{14} is $-\frac{1}{2}ix$ instead of $+x$, as published, was pointed out to the author by Suzanne Davison a year ago and was later confirmed by S. Kumei; The former states that the hand calculation of $[Q_{11},Q_{14}]$ took her two weeks to complete. Note also that v here is ϕ of (7).). The commutator expected is $2Q_1 = 2e^{iv}\{y\partial_x - ix/y\partial_v\}$.

For the computation the explicit value of y ($= \sqrt{1 - x^2}$) must be substituted. The printout follows on the next 6 pages:

* See pp. 44 to 51, Anderson et al. (7).

```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  49 MSEC, COUNT:  70 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
2IEXP(-IT+IV) | ( X(1-X@(2))@(.5)DXDR +(1-X@(2))@(1.5)R@(-1)DXDX
-I(1-X@(2))@(-.5)DRDV +(1-X@(2))@(-.5)R@(-1)DVDV +(1-.5R)
(1-X@(2))@(.5)DR +(.5-2R@(-1))X(1-X@(2))@(.5)DX -.5I(1-X@(2))@(.5)DT
-.5I(1-X@(2))@(-.5)DV -.25R(1-X@(2))@(.5) )
#
2IEXP(IT) | ( -(1-X@(2))DXDR +(X-X@(3))R@(-1)DXDX
+XR@(-1)(1-X@(2))@(-1)DVDV +(1+.5R)XDR
+.5(1-X@(2)-4X@(2)R@(-1))DX -.5IXDT -.25RX )
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (1977 MSEC, 1870 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  14392 MSEC, 22708 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  15750 MSEC, 30215 STATEMENTS.

 ORDERED RESULT:


 +< 8IIEXP(-IT+IV)X(1-X@(2))@(.5)EXP(IT)X -4IIEXP(IT)EXP(-IT+IV)X(1-X@(2))@(-0.5)X
 +4IIEXP(IT)X@(2)EXP(-IT+IV)X(1-X@(2))@(-0.5)X +4IIEXP(IT)EXP(-IT+IV)(1-X@(2))@(.5)
 -4IIEXP(IT)X@(2)EXP(-IT+IV)(1-X@(2))@(.5) >DXDRDR

 +< -4IIIEXP(IT)EXP(-IT+IV)X(1-X@(2))@(-1.5) +4IIIEXP(IT)X@(2)EXP(-IT+IV)X(1-X@(2))@(-1.5) >DVDRDR

 +< 4IIEXP(-IT+IV)X(1-X@(2))@(.5)EXP(IT)R@(-1) -12IIEXP(-IT+IV)X(1-X@(2))@(.5)EXP(IT)R@(-1)X@(2)
 -12IIEXP(IT)EXP(-IT+IV)X(1-X@(2))@(0.5)R@(-1) +12IIEXP(IT)X@(2)EXP(-IT+IV)X(1-X@(2))@(0.5)R@(-1)
 +8IIEXP(-IT+IV)(1-X@(2))@(1.5)R@(-1)EXP(IT)X +8IIEXP(-IT+IV)(1-X@(2))@(1.5)R@(-1)EXP(IT)X
 +4IIEXP(IT)XR@(-1)EXP(-IT+IV)X(1-X@(2))@(-0.5)X -4IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)X(1-X@(2))@(-0.5)X
 -4IIEXP(IT)XR@(-1)EXP(-IT+IV)(1-X@(2))@(.5) +4IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)(1-X@(2))@(.5)
 +4IIEXP(IT)XR@(-1)EXP(-IT+IV)X(1-X@(2))@(-0.5)X -4IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)X(1-X@(2))@(-0.5)X
 -4IIEXP(IT)XR@(-1)EXP(-IT+IV)(1-X@(2))@(.5) +4IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)(1-X@(2))@(.5) >DXDXDR

 +< 4IIIEXP(IT)XR@(-1)EXP(-IT+IV)X(1-X@(2))@(-1.5) -4IIIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)X(1-X@(2))@(-1.5)
 +4IIIEXP(IT)XR@(-1)EXP(-IT+IV)X(1-X@(2))@(-1.5) -4IIIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)X(1-X@(2))@(-1.5)
 -4IIIEXP(IT)XR@(-1)(1-X@(2))@(-1)EXP(-IT+IV)X(1-X@(2))@(.5)
 -4IIIEXP(IT)XR@(-1)(1-X@(2))@(-1)EXP(-IT+IV)X(1-X@(2))@(.5) >DVDXDR

 +< 8IIEXP(-IT+IV)X(1-X@(2))@(.5)EXP(IT)R@(-1)X(1-X@(2))@(-2)X
 +4IIEXP(-IT+IV)X(1-X@(2))@(.5)EXP(IT)R@(-1)(1-X@(2))@(-1) +4IIEXP(IT)EXP(-IT+IV)X(1-X@(2))@(-1.5)R@(-1)
 -4IIEXP(IT)X@(2)EXP(-IT+IV)X(1-X@(2))@(-1.5)R@(-1) +4IIIEXP(IT)XR@(-1)(1-X@(2))@(-1)EXP(-IT+IV)(1-X@(2))@(-.5)
 +4IIIEXP(IT)XR@(-1)(1-X@(2))@(-1)EXP(-IT+IV)(1-X@(2))@(-.5) >DVDVDR

 +< -4IIEXP(-IT+IV)X(1-X@(2))@(.5)EXP(IT)XR@(-2) +4IIEXP(-IT+IV)X(1-X@(2))@(.5)EXP(IT)X@(3)R@(-2)
 -4IIEXP(IT)EXP(-IT+IV)(1-X@(2))@(1.5)R@(-2) +4IIEXP(IT)X@(2)EXP(-IT+IV)(1-X@(2))@(1.5)R@(-2)
 +4IIEXP(-IT+IV)(1-X@(2))@(1.5)R@(-1)EXP(IT)R@(-1) -12IIEXP(-IT+IV)(1-X@(2))@(1.5)R@(-1)EXP(IT)R@(-1)X@(2)
 +4IIEXP(-IT+IV)(1-X@(2))@(1.5)R@(-1)EXP(IT)R@(-1) -12IIEXP(-IT+IV)(1-X@(2))@(1.5)R@(-1)EXP(IT)R@(-1)X@(2)
 +12IIEXP(IT)XR@(-1)EXP(-IT+IV)X(1-X@(2))@(0.5)R@(-1) -12IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)X(1-X@(2))@(0.5)R@(-1)
 +12IIEXP(IT)XR@(-1)EXP(-IT+IV)X(1-X@(2))@(0.5)R@(-1) -12IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)X(1-X@(2))@(0.5)R@(-1) >DXDXDX

 +< 4IIIEXP(-IT+IV)(1-X@(2))@(-.5)EXP(IT)XR@(-2) -4IIIEXP(-IT+IV)(1-X@(2))@(-.5)EXP(IT)X@(3)R@(-2)
 -4IIIEXP(IT)XR@(-1)(1-X@(2))@(-1)EXP(-IT+IV)(1-X@(2))@(1.5)R@(-1)
 -4IIIEXP(IT)XR@(-1)(1-X@(2))@(-1)EXP(-IT+IV)(1-X@(2))@(1.5)R@(-1) >DVDXDX

 +< -4IIEXP(-IT+IV)X(1-X@(2))@(.5)EXP(IT)X(1-X@(2))@(-1)R@(-2) -4IIEXP(IT)EXP(-IT+IV)(1-X@(2))@(-.5)R@(-2)
 +4IIEXP(IT)X@(2)EXP(-IT+IV)(1-X@(2))@(-.5)R@(-2) +8IIEXP(-IT+IV)(1-X@(2))@(1.5)R@(-1)EXP(IT)R@(-1)X(1-X@(2))@(-2)X
```

```
+4IIEXP(-IT+IV)(1-X@(2))@(1.5)R@(-1)EXP(IT)R@(-1)(1-X@(2))@(-1)
+8IIEXP(-IT+IV)(1-X@(2))@(1.5)R@(-1)EXP(IT)R@(-1)X(1-X@(2))@(-2)X
+4IIEXP(-IT+IV)(1-X@(2))@(1.5)R@(-1)EXP(IT)R@(-1)(1-X@(2))@(-1)
-4IIEXP(IT)XR@(-1)EXP(-IT+IV)X(1-X@(2))@(-1.5)R@(-1) +4IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)X(1-X@(2))@(-1.5)R@(-1)
-4IIEXP(IT)XR@(-1)EXP(-IT+IV)X(1-X@(2))@(-1.5)R@(-1) +4IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)X(1-X@(2))@(-1.5)R@(-1) >DVDVDX

+< 4IIIEXP(-IT+IV)(1-X@(2))@(-.5)EXP(IT)X(1-X@(2))@(-1)R@(-2)
-4IIIEXP(IT)XR@(-1)(1-X@(2))@(-1)EXP(-IT+IV)(1-X@(2))@(-.5)R@(-1)
-4IIIEXP(IT)XR@(-1)(1-X@(2))@(-1)EXP(-IT+IV)(1-X@(2))@(-.5)R@(-1) >DVDVDV


+< 4IIEXP(-IT+IV)X(1-X@(2))@(.5)EXP(IT) +2IIEXP(-IT+IV)X(1-X@(2))@(.5)EXP(IT)R
-4IIEXP(IT)EXP(-IT+IV)X(1-X@(2))@(-0.5) +4IIEXP(IT)X@(2)EXP(-IT+IV)X(1-X@(2))@(-0.5)
+2IIEXP(IT)EXP(-IT+IV)X(1-X@(2))@(-0.5)R -2IIEXP(IT)X@(2)EXP(-IT+IV)X(1-X@(2))@(-0.5)R >DRDR

+< 2IIEXP(-IT+IV)X(1-X@(2))@(.5)EXP(IT)X -4IIEXP(-IT+IV)X(1-X@(2))@(.5)EXP(IT)X
-16IIEXP(-IT+IV)X(1-X@(2))@(.5)EXP(IT)R@(-1)X -2IIEXP(IT)EXP(-IT+IV)(1-X@(2))@(.5)
+2IIEXP(IT)X@(2)EXP(-IT+IV)(1-X@(2))@(.5) -2IIEXP(IT)EXP(-IT+IV)X(1-X@(2))@(-0.5)X
+2IIEXP(IT)X@(2)EXP(-IT+IV)X(1-X@(2))@(-0.5)X +2IIEXP(IT)EXP(-IT+IV)(1-X@(2))@(.5)
-2IIEXP(IT)X@(2)EXP(-IT+IV)(1-X@(2))@(.5) +8IIEXP(IT)EXP(-IT+IV)R@(-1)X(1-X@(2))@(-0.5)X
-8IIEXP(IT)X@(2)EXP(-IT+IV)R@(-1)X(1-X@(2))@(-0.5)X -8IIEXP(IT)EXP(-IT+IV)R@(-1)(1-X@(2))@(.5)
+8IIEXP(IT)X@(2)EXP(-IT+IV)R@(-1)(1-X@(2))@(.5) +8IIEXP(-IT+IV)(1-X@(2))@(1.5)R@(-1)EXP(IT)
+4IIEXP(IT)XR@(-1)EXP(-IT+IV)X(1-X@(2))@(-1.5)X@(2) -4IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)X(1-X@(2))@(-1.5)X@(2)
+8IIEXP(IT)XR@(-1)EXP(-IT+IV)(1-X@(2))@(-0.5)X -8IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)(1-X@(2))@(-0.5)X
+4IIEXP(IT)XR@(-1)EXP(-IT+IV)X(1-X@(2))@(-0.5) -4IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)X(1-X@(2))@(-0.5)
+4IIEXP(-IT+IV)(1-X@(2))@(1.5)R@(-1)EXP(IT) +2IIEXP(-IT+IV)(1-X@(2))@(1.5)R@(-1)EXP(IT)R
+4IIEXP(-IT+IV)(1-X@(2))@(1.5)R@(-1)EXP(IT) +2IIEXP(-IT+IV)(1-X@(2))@(1.5)R@(-1)EXP(IT)R
+4IIEXP(IT)XR@(-1)EXP(-IT+IV)X(1-X@(2))@(-0.5) -4IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)X(1-X@(2))@(-0.5)
-2IIEXP(IT)XR@(-1)EXP(-IT+IV)X(1-X@(2))@(-0.5)R +2IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)X(1-X@(2))@(-0.5)R
+4IIEXP(IT)XR@(-1)EXP(-IT+IV)X(1-X@(2))@(-0.5) -4IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)X(1-X@(2))@(-0.5)
-2IIEXP(IT)XR@(-1)EXP(-IT+IV)X(1-X@(2))@(-0.5)R +2IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)X(1-X@(2))@(-0.5)R
-4IIIIEXP(IT)XR@(-1)(1-X@(2))@(-1)EXP(-IT+IV)X(1-X@(2))@(.5) +4IIEXP(-IT+IV)X(1-X@(2))@(.5)EXP(IT)X
-16IIEXP(-IT+IV)R@(-1)X(1-X@(2))@(.5)EXP(IT)X +2IIEXP(IT)EXP(-IT+IV)X(1-X@(2))@(-0.5)X
-2IIEXP(IT)X@(2)EXP(-IT+IV)X(1-X@(2))@(-0.5)X -8IIEXP(IT)X@(2)R@(-1)EXP(-IT+IV)X(1-X@(2))@(-0.5)X
-2IIEXP(IT)EXP(-IT+IV)(1-X@(2))@(.5) +2IIEXP(IT)X@(2)EXP(-IT+IV)(1-X@(2))@(.5)
+8IIEXP(IT)X@(2)R@(-1)EXP(-IT+IV)(1-X@(2))@(.5) +2IIIIEXP(-IT+IV)(1-X@(2))@(.5)EXP(IT)
-2IIIIEXP(-IT+IV)(1-X@(2))@(.5)EXP(IT)X@(2) -2IIIIEXP(IT)XEXP(-IT+IV)X(1-X@(2))@(.5) >DXDR

+< -2IIIEXP(IT)EXP(-IT+IV)X(1-X@(2))@(-1.5) +2IIIEXP(IT)X@(2)EXP(-IT+IV)X(1-X@(2))@(-1.5)
+12IIIEXP(IT)XR@(-1)EXP(-IT+IV)X(1-X@(2))@(-2.5)X -12IIIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)X(1-X@(2))@(-2.5)X
+4IIIEXP(IT)XR@(-1)EXP(-IT+IV)(1-X@(2))@(-1.5) -4IIIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)(1-X@(2))@(-1.5)
-2IIIEXP(-IT+IV)(1-X@(2))@(-.5)EXP(IT)X +4IIIIIEXP(IT)XR@(-1)(1-X@(2))@(-1)EXP(-IT+IV)(1-X@(2))@(-.5)
-4IIIEXP(IT)XR@(-1)(1-X@(2))@(-1)EXP(-IT+IV)(1-X@(2))@(.5)
+2IIIEXP(IT)XR@(-1)(1-X@(2))@(-1)EXP(-IT+IV)R(1-X@(2))@(.5)
-4IIIEXP(IT)XR@(-1)(1-X@(2))@(-1)EXP(-IT+IV)(1-X@(2))@(.5)
+2IIIEXP(IT)XR@(-1)(1-X@(2))@(-1)EXP(-IT+IV)R(1-X@(2))@(.5) +2IIIEXP(IT)EXP(-IT+IV)X(1-X@(2))@(-1.5)
-2IIIEXP(IT)X@(2)EXP(-IT+IV)X(1-X@(2))@(-1.5) -8IIIEXP(IT)X@(2)R@(-1)EXP(-IT+IV)X(1-X@(2))@(-1.5)
+2IIIIIEXP(IT)XEXP(-IT+IV)(1-X@(2))@(-.5) >DVDR

+< -2IIIEXP(-IT+IV)X(1-X@(2))@(.5)EXP(IT) +2IIIEXP(IT)EXP(-IT+IV)X(1-X@(2))@(-0.5)
-2IIIEXP(IT)X@(2)EXP(-IT+IV)X(1-X@(2))@(-0.5) >DTDR

+< -4IIEXP(-IT+IV)X(1-X@(2))@(.5)EXP(IT)R@(-2) +12IIEXP(-IT+IV)X(1-X@(2))@(.5)EXP(IT)X@(2)R@(-2)
+12IIEXP(IT)EXP(-IT+IV)X(1-X@(2))@(0.5)R@(-2) -12IIEXP(IT)X@(2)EXP(-IT+IV)X(1-X@(2))@(0.5)R@(-2)
+8IIEXP(-IT+IV)X(1-X@(2))@(.5)EXP(IT)X@(2)R@(-2) +8IIEXP(IT)EXP(-IT+IV)X(1-X@(2))@(.5)R@(-2)
-8IIEXP(IT)X@(2)EXP(-IT+IV)X(1-X@(2))@(.5)R@(-2) -24IIEXP(-IT+IV)(1-X@(2))@(1.5)R@(-1)EXP(IT)R@(-1)X
-12IIEXP(IT)XR@(-1)EXP(-IT+IV)X(1-X@(2))@(-0.5)R@(-1)X +12IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)X(1-X@(2))@(-0.5)R@(-1)X
+12IIEXP(IT)XR@(-1)EXP(-IT+IV)(1-X@(2))@(0.5)R@(-1) -12IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)(1-X@(2))@(0.5)R@(-1)
-4IIEXP(-IT+IV)(1-X@(2))@(1.5)R@(-1)EXP(IT)X -16IIEXP(-IT+IV)(1-X@(2))@(1.5)R@(-1)EXP(IT)R@(-1)X
-4IIEXP(-IT+IV)(1-X@(2))@(1.5)R@(-1)EXP(IT)X -16IIEXP(-IT+IV)(1-X@(2))@(1.5)R@(-1)EXP(IT)R@(-1)X
```

```
+2IIEXP(IT)XR@(-1)EXP(-IT+IV)X(1-X@(2))@(-0.5)X -2IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)X(1-X@(2))@(-0.5)X
-2IIEXP(IT)XR@(-1)EXP(-IT+IV)(1-X@(2))@(.5) +2IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)(1-X@(2))@(.5)
-8IIEXP(IT)XR@(-1)EXP(-IT+IV)R@(-1)X(1-X@(2))@(-0.5)X +8IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)R@(-1)X(1-X@(2))@(-0.5)X
+8IIEXP(IT)XR@(-1)EXP(-IT+IV)R@(-1)(1-X@(2))@(.5) -8IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)R@(-1)(1-X@(2))@(.5)
+2IIEXP(IT)XR@(-1)EXP(-IT+IV)X(1-X@(2))@(-0.5)X -2IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)X(1-X@(2))@(-0.5)X
-2IIEXP(IT)XR@(-1)EXP(-IT+IV)(1-X@(2))@(.5) +2IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)(1-X@(2))@(.5)
-8IIEXP(IT)XR@(-1)EXP(-IT+IV)R@(-1)X(1-X@(2))@(-0.5)X +8IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)R@(-1)X(1-X@(2))@(-0.5)X
+8IIEXP(IT)XR@(-1)EXP(-IT+IV)R@(-1)(1-X@(2))@(.5) -8IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)R@(-1)(1-X@(2))@(.5)
-4IIIIEXP(IT)XR@(-1)(1-X@(2))@(-1)EXP(-IT+IV)(1-X@(2))@(1.5)R@(-1) +4IIEXP(IT)XEXP(-IT+IV)(1-X@(2))@(1.5)R@(-2)
+2IIEXP(IT)RXEXP(-IT+IV)(1-X@(2))@(1.5)R@(-2) -4IIEXP(-IT+IV)(1-X@(2))@(.5)EXP(IT)XR@(-2)
+2IIEXP(-IT+IV)R(1-X@(2))@(.5)EXP(IT)XR@(-2) +4IIEXP(-IT+IV)(1-X@(2))@(.5)EXP(IT)X@(3)R@(-2)
-2IIEXP(-IT+IV)R(1-X@(2))@(.5)EXP(IT)X@(3)R@(-2) +2IIEXP(-IT+IV)X(1-X@(2))@(.5)EXP(IT)R@(-1)
-8IIEXP(-IT+IV)R@(-1)X(1-X@(2))@(.5)EXP(IT)R@(-1) -6IIEXP(-IT+IV)X(1-X@(2))@(.5)EXP(IT)R@(-1)X@(2)
+24IIEXP(-IT+IV)R@(-1)X(1-X@(2))@(.5)EXP(IT)R@(-1)X@(2) +6IIEXP(IT)EXP(-IT+IV)X(1-X@(2))@(0.5)R@(-1)
-6IIEXP(IT)X@(2)EXP(-IT+IV)X(1-X@(2))@(0.5)R@(-1) -24IIEXP(IT)X@(2)R@(-1)EXP(-IT+IV)X(1-X@(2))@(0.5)R@(-1)
-2IIIIEXP(-IT+IV)(1-X@(2))@(.5)EXP(IT)XR@(-1) +2IIIIEXP(-IT+IV)(1-X@(2))@(.5)EXP(IT)X@(3)R@(-1)
-2IIIIEXP(IT)XEXP(-IT+IV)(1-X@(2))@(1.5)R@(-1) >DXDX

+< 2IIIEXP(IT)XR@(-1)EXP(-IT+IV)X(1-X@(2))@(-1.5) -2IIIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)X(1-X@(2))@(-1.5)
+2IIIEXP(IT)XR@(-1)EXP(-IT+IV)X(1-X@(2))@(-1.5) -2IIIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)X(1-X@(2))@(-1.5)
-8IIIEXP(-IT+IV)(1-X@(2))@(-.5)EXP(IT)X@(2)R@(-2) -2IIIEXP(IT)XR@(-1)(1-X@(2))@(-1)EXP(-IT+IV)X(1-X@(2))@(.5)
+8IIIEXP(IT)XR@(-1)(1-X@(2))@(-1)EXP(-IT+IV)R@(-1)X(1-X@(2))@(.5)
-2IIIEXP(IT)XR@(-1)(1-X@(2))@(-1)EXP(-IT+IV)X(1-X@(2))@(.5)
+8IIIEXP(IT)XR@(-1)(1-X@(2))@(-1)EXP(-IT+IV)R@(-1)X(1-X@(2))@(.5) >DVDX

+< -2IIIEXP(-IT+IV)(1-X@(2))@(1.5)R@(-1)EXP(IT) -2IIIEXP(-IT+IV)(1-X@(2))@(1.5)R@(-1)EXP(IT)
-2IIIEXP(IT)XR@(-1)EXP(-IT+IV)X(1-X@(2))@(-0.5) +2IIIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)X(1-X@(2))@(-0.5)
-2IIIEXP(IT)XR@(-1)EXP(-IT+IV)X(1-X@(2))@(-0.5) +2IIIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)X(1-X@(2))@(-0.5) >DTDX

+< -8IIEXP(-IT+IV)X(1-X@(2))@(.5)EXP(IT)X(1-X@(2))@(-2)XR@(-2)
-4IIEXP(-IT+IV)X(1-X@(2))@(.5)EXP(IT)(1-X@(2))@(-1)R@(-2) -4IIEXP(IT)EXP(-IT+IV)X(1-X@(2))@(-1.5)R@(-2)
+4IIEXP(IT)X@(2)EXP(-IT+IV)X(1-X@(2))@(-1.5)R@(-2)
+32IIEXP(-IT+IV)(1-X@(2))@(1.5)R@(-1)EXP(IT)R@(-1)X(1-X@(2))@(-3)X@(2)
+16IIEXP(-IT+IV)(1-X@(2))@(1.5)R@(-1)EXP(IT)R@(-1)(1-X@(2))@(-2)X
+8IIEXP(-IT+IV)(1-X@(2))@(1.5)R@(-1)EXP(IT)R@(-1)X(1-X@(2))@(-2)
-12IIEXP(IT)XR@(-1)EXP(-IT+IV)X(1-X@(2))@(-2.5)R@(-1)X +12IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)X(1-X@(2))@(-2.5)R@(-1)X
-4IIEXP(IT)XR@(-1)EXP(-IT+IV)(1-X@(2))@(-1.5)R@(-1) +4IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)(1-X@(2))@(-1.5)R@(-1)
-4IIIIEXP(IT)XR@(-1)(1-X@(2))@(-1)EXP(-IT+IV)(1-X@(2))@(-.5)R@(-1)
+2IIIIEXP(IT)XR@(-1)(1-X@(2))@(-1)EXP(-IT+IV)(1-X@(2))@(-.5)
+2IIIIEXP(IT)XR@(-1)(1-X@(2))@(-1)EXP(-IT+IV)(1-X@(2))@(-.5)
-4IIEXP(-IT+IV)(1-X@(2))@(.5)EXP(IT)X(1-X@(2))@(-1)R@(-2)
+2IIEXP(-IT+IV)R(1-X@(2))@(.5)EXP(IT)X(1-X@(2))@(-1)R@(-2) +4IIEXP(IT)XEXP(-IT+IV)(1-X@(2))@(-.5)R@(-2)
+2IIEXP(IT)RXEXP(-IT+IV)(1-X@(2))@(-.5)R@(-2) +4IIEXP(-IT+IV)X(1-X@(2))@(.5)EXP(IT)R@(-1)X(1-X@(2))@(-2)X
-16IIEXP(-IT+IV)R@(-1)X(1-X@(2))@(.5)EXP(IT)R@(-1)X(1-X@(2))@(-2)X
+2IIEXP(-IT+IV)X(1-X@(2))@(.5)EXP(IT)R@(-1)(1-X@(2))@(-1)
-8IIEXP(-IT+IV)R@(-1)X(1-X@(2))@(.5)EXP(IT)R@(-1)(1-X@(2))@(-1) -2IIEXP(IT)EXP(-IT+IV)X(1-X@(2))@(-1.5)R@(-1)
+2IIEXP(IT)X@(2)EXP(-IT+IV)X(1-X@(2))@(-1.5)R@(-1) +8IIEXP(IT)X@(2)R@(-1)EXP(-IT+IV)X(1-X@(2))@(-1.5)R@(-1)
-2IIIIEXP(-IT+IV)(1-X@(2))@(.5)EXP(IT)XR@(-1)(1-X@(2))@(-1) -2IIIIEXP(IT)XEXP(-IT+IV)(1-X@(2))@(-.5)R@(-1) >DVDV

+< 2IIIIEXP(IT)XR@(-1)(1-X@(2))@(-1)EXP(-IT+IV)(1-X@(2))@(.5)
+2IIIIEXP(IT)XR@(-1)(1-X@(2))@(-1)EXP(-IT+IV)(1-X@(2))@(.5) >DTDV


+< 2IIEXP(-IT+IV)X(1-X@(2))@(.5)EXP(IT) +2IIEXP(IT)EXP(-IT+IV)X(1-X@(2))@(-0.5)
-2IIEXP(IT)X@(2)EXP(-IT+IV)X(1-X@(2))@(-0.5) -IIEXP(-IT+IV)X(1-X@(2))@(.5)EXP(IT)R
+IIEXP(IT)EXP(-IT+IV)X(1-X@(2))@(-0.5)R -IIEXP(IT)X@(2)EXP(-IT+IV)X(1-X@(2))@(-0.5)R
+4IIEXP(IT)XR@(-1)EXP(-IT+IV)X(1-X@(2))@(-1.5)X -4IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)X(1-X@(2))@(-1.5)X
+4IIEXP(IT)XR@(-1)EXP(-IT+IV)(1-X@(2))@(-0.5) -4IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)(1-X@(2))@(-0.5)
-2IIEXP(IT)XR@(-1)EXP(-IT+IV)X(1-X@(2))@(-1.5)RX +2IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)X(1-X@(2))@(-1.5)RX
-2IIEXP(IT)XR@(-1)EXP(-IT+IV)(1-X@(2))@(-0.5)R +2IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)(1-X@(2))@(-0.5)R
```

```
-4IIIIEXP(IT)XR@(-1)(1-X@(2))@(-1)EXP(-IT+IV)(1-X@(2))@(.5)
+2IIIIEXP(IT)XP@(-1)(1-X@(2))@(-1)EXP(-IT+IV)R(1-X@(2))@(.5) +2IIEXP(-IT+IV)(1-X@(2))@(.5)EXP(IT)X
-IIEXP(-IT+IV)R(1-X@(2))@(.5)EXP(IT)X +2IIEXP(IT)XEXP(-IT+IV)(1-X@(2))@(.5) +IIEXP(IT)RXEXP(-IT+IV)(1-X@(2))@(.5)
+2IIEXP(-IT+IV)X(1-X@(2))@(.5)EXP(IT) -8IIEXP(-IT+IV)R@(-1)X(1-X@(2))@(.5)EXP(IT)
+IIEXP(-IT+IV)X(1-X@(2))@(.5)EXP(IT)R -4IIEXP(-IT+IV)R@(-1)X(1-X@(2))@(.5)EXP(IT)R
+2IIEXP(IT)EXP(-IT+IV)X(1-X@(2))@(-0.5) -2IIEXP(IT)X@(2)EXP(-IT+IV)X(1-X@(2))@(-0.5)
-8IIEXP(IT)X@(2)R@(-1)EXP(-IT+IV)X(1-X@(2))@(-0.5) -IIEXP(IT)EXP(-IT+IV)X(1-X@(2))@(-0.5)R
+IIEXP(IT)X@(2)EXP(-IT+IV)X(1-X@(2))@(-0.5)R +4IIEXP(IT)X@(2)R@(-1)EXP(-IT+IV)X(1-X@(2))@(-0.5)R
-2IIIIEXP(-IT+IV)(1-X@(2))@(.5)EXP(IT)X -IIIIEXP(-IT+IV)(1-X@(2))@(.5)EXP(IT)RX
-2IIIIEXP(IT)XEXP(-IT+IV)(1-X@(2))@(.5) +IIIIEXP(IT)XEXP(-IT+IV)R(1-X@(2))@(.5) >DR

+< 16IIEXP(-IT+IV)X(1-X@(2))@(.5)EXP(IT)XR@(-2) -8IIEXP(IT)EXP(-IT+IV)X(1-X@(2))@(-0.5)XR@(-2)
+8IIEXP(IT)X@(2)EXP(-IT+IV)X(1-X@(2))@(-0.5)XR@(-2) +8IIEXP(IT)EXP(-IT+IV)(1-X@(2))@(.5)R@(-2)
-8IIEXP(IT)X@(2)EXP(-IT+IV)(1-X@(2))@(.5)R@(-2) -IIEXP(-IT+IV)X(1-X@(2))@(.5)EXP(IT)X
-IIEXP(IT)EXP(-IT+IV)(1-X@(2))@(.5) +IIEXP(IT)X@(2)EXP(-IT+IV)(1-X@(2))@(.5)
-4IIEXP(-IT+IV)(1-X@(2))@(1.5)R@(-1)EXP(IT) -16IIEXP(-IT+IV)(1-X@(2))@(1.5)R@(-1)EXP(IT)R@(-1)
-IIEXP(-IT+IV)(1-X@(2))@(1.5)R@(-1)EXP(IT)R -IIEXP(-IT+IV)(1-X@(2))@(1.5)R@(-1)EXP(IT)R
+2IIEXP(IT)XR@(-1)EXP(-IT+IV)X(1-X@(2))@(-1.5)X@(2) -2IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)X(1-X@(2))@(-1.5)X@(2)
+4IIEXP(IT)XR@(-1)EXP(-IT+IV)(1-X@(2))@(-0.5)X -4IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)(1-X@(2))@(-0.5)X
+2IIEXP(IT)XR@(-1)EXP(-IT+IV)X(1-X@(2))@(-0.5) -2IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)X(1-X@(2))@(-0.5)
-8IIEXP(IT)XR@(-1)EXP(-IT+IV)R@(-1)X(1-X@(2))@(-1.5)X@(2)
+8IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)R@(-1)X(1-X@(2))@(-1.5)X@(2) -16IIEXP(IT)XR@(-1)EXP(-IT+IV)R@(-1)(1-X@(2))@(-0.5)X
+16IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)R@(-1)(1-X@(2))@(-0.5)X -8IIEXP(IT)XR@(-1)EXP(-IT+IV)R@(-1)X(1-X@(2))@(-0.5)
+8IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)R@(-1)X(1-X@(2))@(-0.5) -IIEXP(IT)XR@(-1)EXP(-IT+IV)X(1-X@(2))@(-0.5)R
+IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)X(1-X@(2))@(-0.5)R -IIEXP(IT)XR@(-1)EXP(-IT+IV)X(1-X@(2))@(-0.5)R
+IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)X(1-X@(2))@(-0.5)R -2IIIIEXP(IT)XR@(-1)(1-X@(2))@(-1)EXP(-IT+IV)X(1-X@(2))@(.5)
+8IIIIEXP(IT)XR@(-1)(1-X@(2))@(-1)EXP(-IT+IV)R@(-1)X(1-X@(2))@(.5) +8IIEXP(-IT+IV)(1-X@(2))@(.5)EXP(IT)X@(2)R@(-2)
-4IIEXP(-IT+IV)R(1-X@(2))@(.5)EXP(IT)X@(2)R@(-2) -8IIEXP(IT)XEXP(-IT+IV)X(1-X@(2))@(.5)R@(-2)
-4IIEXP(IT)PXEXP(-IT+IV)X(1-X@(2))@(.5)R@(-2) -2IIEXP(-IT+IV)X(1-X@(2))@(.5)EXP(IT)X
+8IIEXP(-IT+IV)R@(-1)X(1-X@(2))@(.5)EXP(IT)X -8IIEXP(-IT+IV)X(1-X@(2))@(.5)EXP(IT)R@(-1)X
+32IIEXP(-IT+IV)R@(-1)X(1-X@(2))@(.5)EXP(IT)R@(-1)X +IIEXP(IT)EXP(-IT+IV)X(1-X@(2))@(-0.5)X
-IIEXP(IT)X@(2)EXP(-IT+IV)X(1-X@(2))@(-0.5)X -4IIEXP(IT)X@(2)R@(-1)EXP(-IT+IV)X(1-X@(2))@(-0.5)X
-IIEXP(IT)EXP(-IT+IV)(1-X@(2))@(.5) +IIEXP(IT)X@(2)EXP(-IT+IV)(1-X@(2))@(.5)
+4IIEXP(IT)X@(2)P@(-1)EXP(-IT+IV)(1-X@(2))@(.5) -4IIEXP(IT)EXP(-IT+IV)R@(-1)X(1-X@(2))@(-0.5)X
+4IIEXP(IT)X@(2)EXP(-IT+IV)R@(-1)X(1-X@(2))@(-0.5)X +16IIEXP(IT)X@(2)R@(-1)EXP(-IT+IV)R@(-1)X(1-X@(2))@(-0.5)X
+4IIEXP(IT)EXP(-IT+IV)R@(-1)(1-X@(2))@(.5) -4IIEXP(IT)X@(2)EXP(-IT+IV)R@(-1)(1-X@(2))@(.5)
-16IIEXP(IT)X@(2)R@(-1)EXP(-IT+IV)R@(-1)(1-X@(2))@(.5) -IIIIEXP(-IT+IV)(1-X@(2))@(.5)EXP(IT)
+IIIIEXP(-IT+IV)(1-X@(2))@(.5)EXP(IT)X@(2) +4IIIIEXP(-IT+IV)(1-X@(2))@(.5)EXP(IT)X@(2)R@(-1)
-IIIIEXP(IT)XEXP(-IT+IV)X(1-X@(2))@(.5) +4IIIIEXP(IT)XEXP(-IT+IV)R@(-1)X(1-X@(2))@(.5) >DX

+< 6IIIEXP(IT)XR@(-1)EXP(-IT+IV)X(1-X@(2))@(-2.5)X -6IIIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)X(1-X@(2))@(-2.5)X
+2IIIEXP(IT)XR@(-1)EXP(-IT+IV)(1-X@(2))@(-1.5) -2IIIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)(1-X@(2))@(-1.5)
+IIIEXP(-IT+IV)(1-X@(2))@(-.5)EXP(IT)X +2IIIIIEXP(IT)XR@(-1)(1-X@(2))@(-1)EXP(-IT+IV)(1-X@(2))@(-.5)
+IIIEXP(IT)XR@(-1)(1-X@(2))@(-1)EXP(-IT+IV)R(1-X@(2))@(.5)
+IIIEXP(IT)XR@(-1)(1-X@(2))@(-1)EXP(-IT+IV)R(1-X@(2))@(.5) +IIIEXP(IT)EXP(-IT+IV)X(1-X@(2))@(-1.5)
-IIIEXP(IT)X@(2)EXP(-IT+IV)X(1-X@(2))@(-1.5) -4IIIEXP(IT)X@(2)R@(-1)EXP(-IT+IV)X(1-X@(2))@(-1.5)
+IIIIIEXP(IT)XEXP(-IT+IV)(1-X@(2))@(-.5) >DV

+< -2IIIEXP(IT)XR@(-1)EXP(-IT+IV)X(1-X@(2))@(-1.5)X +2IIIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)X(1-X@(2))@(-1.5)X
-2IIIEXP(IT)XR@(-1)EXP(-IT+IV)(1-X@(2))@(-0.5) +2IIIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)(1-X@(2))@(-0.5)
+2IIIIIEXP(IT)XR@(-1)(1-X@(2))@(-1)EXP(-IT+IV)(1-X@(2))@(.5) -IIIEXP(-IT+IV)X(1-X@(2))@(.5)EXP(IT)
+4IIIEXP(-IT+IV)R@(-1)X(1-X@(2))@(.5)EXP(IT) -IIIEXP(IT)EXP(-IT+IV)X(1-X@(2))@(-0.5)
+IIIEXP(IT)X@(2)EXP(-IT+IV)X(1-X@(2))@(-0.5) +4IIIEXP(IT)X@(2)R@(-1)EXP(-IT+IV)X(1-X@(2))@(-0.5)
+IIIIIEXP(-IT+IV)(1-X@(2))@(.5)EXP(IT)X +IIIIIEXP(IT)XEXP(-IT+IV)(1-X@(2))@(.5) >DT


+< -IIEXP(-IT+IV)X(1-X@(2))@(.5)EXP(IT) +IIEXP(IT)EXP(-IT+IV)X(1-X@(2))@(-0.5)
-IIEXP(IT)X@(2)EXP(-IT+IV)X(1-X@(2))@(-0.5) -IIEXP(IT)XR@(-1)EXP(-IT+IV)X(1-X@(2))@(-1.5)RX
+IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)X(1-X@(2))@(-1.5)RX -IIEXP(IT)XR@(-1)EXP(-IT+IV)(1-X@(2))@(-0.5)R
+IIEXP(IT)X@(3)R@(-1)EXP(-IT+IV)(1-X@(2))@(-0.5)R +IIIIEXP(IT)XR@(-1)(1-X@(2))@(-1)EXP(-IT+IV)R(1-X@(2))@(.5)
```

```
-IIEXP(-IT+IV)(1-X@(2))@(.5)EXP(IT)X +0.5IIEXP(-IT+IV)R(1-X@(2))@(.5)EXP(IT)X +IIEXP(IT)XEXP(-IT+IV)(1-X@(2))@(.5)
+0.5IIEXP(IT)RXEXP(-IT+IV)(1-X@(2))@(.5) -0.5IIEXP(-IT+IV)X(1-X@(2))@(.5)EXP(IT)R
+2IIEXP(-IT+IV)P@(-1)X(1-X@(2))@(.5)EXP(IT)R -0.5IIEXP(IT)EXP(-IT+IV)X(1-X@(2))@(-0.5)R
+0.5IIEXP(IT)X@(2)EXP(-IT+IV)X(1-X@(2))@(-0.5)R +2IIEXP(IT)X@(2)P@(-1)EXP(-IT+IV)X(1-X@(2))@(-0.5)R
+0.5IIIIEXP(-IT+IV)(1-X@(2))@(.5)EXP(IT)RX +0.5IIIIEXP(IT)XEXP(-IT+IV)R(1-X@(2))@(.5) >


TIME AND COUNT AT END OF OUTPUT ROUTINE:  17385 MSEC, 33509 STATEMENTS.
TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  40766 MSEC, 51453 STATEMENTS.

SIMPLIFIED RESULT:

+< -4X@(2)EXP(-IT+IV)EXP(IT)(1-X@(2))@(0.5) +4X@(2)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5)
-4X@(4)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5) -4EXP(-IT+IV)EXP(IT)(1-X@(2))@(0.5) >DXDRDR

+< 4IXEXP(-IT+IV)EXP(IT)(1-X@(2))@(-1.5) -4IX@(3)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-1.5) >DVDRDR

+< 16XR@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(0.5) -8X@(3)R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(0.5)
-8X@(3)R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5) +8X@(5)R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5)
-16XR@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(1.5) >DXDXDR

+< -8IX@(2)R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-1.5) +8IX@(4)R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-1.5)
+8IX@(2)R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5) >DVDXDR

+< -4X@(3)R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-1.5) -4XR@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5)
+4XR@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-1.5) >DVDVDR

+< -20X@(2)R@(-2)EXP(-IT+IV)EXP(IT)(1-X@(2))@(0.5) +20X@(4)R@(-2)EXP(-IT+IV)EXP(IT)(1-X@(2))@(0.5)
-4R@(-2)EXP(-IT+IV)EXP(IT)(1-X@(2))@(1.5) +20X@(2)R@(-2)EXP(-IT+IV)EXP(IT)(1-X@(2))@(1.5) >DXDXDX

+< -4IXR@(-2)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5) +4IX@(3)R@(-2)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5)
+8IXR@(-2)EXP(-IT+IV)EXP(IT)(1-X@(2))@(0.5) >DVDXDX

+< -16X@(2)R@(-2)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5) +4R@(-2)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5)
-8X@(4)R@(-2)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-1.5) +8X@(2)R@(-2)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-1.5)
-8R@(-2)EXP(-IT+IV)EXP(IT)(1-X@(2))@(0.5) >DVDVDX

+< 4IXR@(-2)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-1.5) >DVDVDV

+< -4XEXP(-IT+IV)EXP(IT)(1-X@(2))@(0.5) -2XREXP(-IT+IV)EXP(IT)(1-X@(2))@(0.5) +4XEXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5)
-4X@(3)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5) -2XREXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5)
+2X@(3)REXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5) >DRDR

+< -8X@(2)EXP(-IT+IV)EXP(IT)(1-X@(2))@(0.5) +4EXP(-IT+IV)EXP(IT)(1-X@(2))@(0.5)
+16X@(2)R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(0.5) -4X@(4)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5)
+36X@(4)R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5) +4X@(2)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5)
-16R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(1.5) -4EXP(-IT+IV)EXP(IT)(1-X@(2))@(1.5)
-32X@(2)R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5) +4X@(6)R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-1.5)
-4X@(4)R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-1.5) +8R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(0.5) >DXDR

+< 12IX@(3)R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-1.5) +8IXR@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5)
-12IX@(3)R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-2.5) +12IX@(5)R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-2.5) >DVDR

+< 2IXEXP(-IT+IV)EXP(IT)(1-X@(2))@(0.5) -2IXEXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5)
+2IX@(3)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5) >DTDR

+< -36XR@(-2)EXP(-IT+IV)EXP(IT)(1-X@(2))@(0.5) +24X@(3)R@(-2)EXP(-IT+IV)EXP(IT)(1-X@(2))@(0.5)
```

$y=\sqrt{1-x^2}$, $\varphi = v$

$(4e^{i\varphi}/\sqrt{1-x^2})\left[-x^2(1-x^2)+x^2-x^4-(1-x^2)\right]\partial_{xrr} = -4e^{i\varphi}y\,\partial_{xrr}$

$\frac{4ie^{i\varphi}x}{y}\partial_{\varphi rr}$

$(8e^{i\varphi}/r\sqrt{1-x^2})\left[2x(1-x^2)-x^3(1-x^2)-x^3+x^5-2x(1-x^2)^2\right]\partial_{xxr}$

$(8ix^2e^{i\varphi}/r\sqrt{(1-x^2)^3})\left[-1+x^2+(1-x^2)\right]\partial_{\varphi xr}$

$(4xe^{i\varphi}/r\sqrt{(1-x^2)^3})\left[-x^2-(1-x^2)+1\right]\partial_{\varphi\varphi r}$

$(4e^{i\varphi}\sqrt{1-x^2}/r^2)\left[-5x^2+5x^4-(1-x^2)+5x^2(1-x^2)\right]\partial_{xxx} = \frac{-4e^{i\varphi}y^3}{r^2}\partial_{xxx}$

$(4ixe^{i\varphi}/r^2\sqrt{1-x^2})\left[-1+x^2+2(1-x^2)\right]\partial_{\varphi xx} = \frac{4ie^{i\varphi}xy}{r^2}\partial_{\varphi xx}$

$(4e^{i\varphi}/r^2\sqrt{(1-x^2)^3})\left[-4x^2(1-x^2)+(1-x^2)-2x^4+2x^2-2(1-x^2)^2\right]\partial_{\varphi\varphi x} = \frac{-4e^{i\varphi}}{r^2y}\partial_{\varphi\varphi x}$

$\frac{4ixe^{i\varphi}}{r^2y^3}\partial_{\varphi\varphi\varphi}$

$(2e^{i\varphi}x/\sqrt{1-x^2})\left[-2(1-x^2)-r(1-x^2)+2-2x^2-r+x^2r\right]\partial_{rr} = -4xry\partial_{rr}$

$(4e^{i\varphi}/\sqrt{(1-x^2)^3})\left[-2x^2(1-x^2)^2+(1-x^2)^2+4\frac{x^2}{v}(1-x^2)^2-x^4(1-x^2)+9\frac{x^4}{v}(1-x^2)+x^2(1-x^2)-4\frac{(1-x^2)^3}{v}-(1-x^2)^3-8\frac{x^2}{v}(1-x^2)+\frac{x^6}{v}-\frac{x^4}{v}+\frac{2}{v}(1-x^2)^2\right]\partial_{xr} = -\frac{8y}{v}e^{i\varphi}\partial_{xr}$

65

```
+4XP@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(1.5) +12X@(3)R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(0.5)
-8XP@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(0.5) +28X@(3)R@(-2)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5)
+52XR@(-2)EXP(-IT+IV)EXP(IT)(1-X@(2))@(1.5) -4X@(3)R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5)
+4X@(5)R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5) -28X@(5)R@(-2)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5) >DXDX

+< -4IX@(2)R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-1.5) +4IX@(4)R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-1.5)
-8IX@(2)R@(-2)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5) +4IX@(2)R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5) >DVDX

+< 4IR@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(1.5) -4IX@(4)R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5)
+4IX@(2)R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5) >DTDX

+< -24X@(3)R@(-2)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-1.5) -12XR@(-2)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5)
+4XP@(-2)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-1.5) -10XR@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5)
-6X@(3)R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-1.5) +6XR@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-1.5)
-12X@(5)R@(-2)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-2.5) +12X@(3)R@(-2)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-2.5) >DVDV

+< 4XR@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5) >DTDV


+< -8XEXP(-IT+IV)EXP(IT)(1-X@(2))@(0.5) -2X@(3)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5)
-8XP@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5) +12X@(3)R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5)
-2X@(5)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-1.5) +8XR@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(0.5)
-4X@(3)R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-1.5) +4X@(5)R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-1.5)
+2X@(3)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-1.5) >DR

+< -24X@(2)R@(-2)EXP(-IT+IV)EXP(IT)(1-X@(2))@(0.5) +40X@(2)R@(-2)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5)
-48X@(4)R@(-2)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5) -8R@(-2)EXP(-IT+IV)EXP(IT)(1-X@(2))@(0.5)
+16X@(2)R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(0.5) +X@(2)EXP(-IT+IV)EXP(IT)(1-X@(2))@(0.5)
+EXP(-IT+IV)EXP(IT)(1-X@(2))@(0.5) +X@(2)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5)
+4R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(1.5) +16R@(-2)EXP(-IT+IV)EXP(IT)(1-X@(2))@(1.5)
+2EXP(-IT+IV)EXP(IT)(1-X@(2))@(1.5) -X@(4)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5)
-2X@(4)R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-1.5) +2X@(6)R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-1.5)
-4X@(2)R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5) +6X@(4)R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5)
-4R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(0.5) -8X@(6)R@(-2)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-1.5)
+8X@(4)R@(-2)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-1.5) >DX

+< -6IX@(3)R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-2.5) +6IX@(5)R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-2.5)
+6IX@(3)R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-1.5) +IX@(3)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-1.5)
-2IXEXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5) -IXEXP(-IT+IV)EXP(IT)(1-X@(2))@(-1.5) >DV

+< 2IX@(3)R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-1.5) -2IX@(5)R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-1.5)
+4IXR@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5) -6IX@(3)R@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5)
+3IXEXP(-IT+IV)EXP(IT)(1-X@(2))@(0.5) -IX@(3)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5)
-4IXR@(-1)EXP(-IT+IV)EXP(IT)(1-X@(2))@(0.5) +IXEXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5) >DT


+< -XEXP(-IT+IV)EXP(IT)(1-X@(2))@(0.5) +XEXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5) -2X@(3)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5)
+X@(3)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-1.5) -X@(5)EXP(-IT+IV)EXP(IT)(1-X@(2))@(-1.5)
-0.5X@(3)REXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5) +0.5XREXP(-IT+IV)EXP(IT)(1-X@(2))@(0.5)
+0.5XREXP(-IT+IV)EXP(IT)(1-X@(2))@(-0.5) >
```



```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  41694 MSEC, COUNT:  52811 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
TERM
```

The compilation and termination statistics are as follows:

```
COMPILATION TIME  6.271  SECONDS  (including listing)
MEMORY USAGE (DECIMAL BYTES)
CODE:              93114
STRINGS:            4765
VARIABLES:         14432
CONSTANTS:          3800
TOTAL:            116111
AVAILABLE:        344329

NORMAL TERMINATION IN STATEMENT     1007
NUMBER OF STATEMENTS EXECUTED      52815
EXECUTION TIME (SECONDS)          41.702
MSEC/STATEMENT                     0.789
NUMBER OF STORAGE REGENERATIONS       39
```

Note that the program was set to run with a larger than usual amount of dynamic memory available; this was accomplished by changing the job control language (see Section 6.3, p. 129, for further details). The improvement in readability of the "simplified result", as compared to the "ordered result", is due to the cannonical ordering of factors within additive components of functions (discussed further in Section 5.2, p. 89) and the summation of common terms; the possibility of even further improvement along the lines suggested by the factorization performed in the margin of the printout is the subject of Section 6.4, p. 132.

Proceeding as indicated on the printout, we arrive at the result: $A = e^{iv}\{-4y\partial_x\partial_r\partial_r + 4ix/y\partial_v\partial_r\partial_r - 4y^3/r^2\partial_x\partial_x\partial_x + 4ixy/r^2\partial_v\partial_x\partial_x - 4/r^2y\partial_v\partial_v\partial_x + 4ix/r^2y^3\partial_v\partial_v\partial_v - 4xry\partial_r\partial_r - 8y/r\partial_x\partial_r + 8ix/ry\partial_v\partial_r + (16xy/r^2 - 4xy^3/r)\partial_x\partial_x - 8ix^2/r^2y\partial_v\partial_x + 4iy/r\partial_t\partial_x - (8x/r^2y^3 + 4x/ry)\partial_v\partial_v + 4x/ry\partial_t\partial_v - 8xy\partial_r + (8/r^2 + 8x^2/r + 3)y\partial_x - 3ix/y\partial_v + 4ixy\partial_t + xry\}$.

Since $K\Psi = -2D(H - i\partial_t)D^{-1}\Psi = 0$, where $K = \partial_r\partial_r + 2/r\partial_r + y^2/r^2\partial_x\partial_x - 2x/r^2\partial_x + 1/r^2y^2\partial_v\partial_v - i/r\partial_t - \frac{1}{4}$ and is a solution, $-y\partial_x K\Psi = 0$, $ix/y\partial_v K\Psi = 0$, and $-xryK\Psi = 0$. Thus we see that $A\Psi = 2e^{iv}\{y\partial_x - ix/y\partial_v\}\Psi$ and that the commutator is the expected result*.

Example 5. In the other examples there were not many constants in the operators. The operators:

$$L = \exp((g_1 + ih)x)u(\partial_x - (g_1 - ih)u\partial_u - (g_2 - ih)v\partial_v)$$

$$M = \exp((g_1 - ih)x)u(\partial_x - (g_1 + ih)u\partial_u - (g_2 + ih)v\partial_v)$$

$$N = \exp(-2ihx)(\partial_x - (g_1 + ih)u\partial_u - (g_2 + ih)v\partial_v)$$

form a closed subalgebra:

[↓ , →]	L	M
M	0	-
N	2ihM	0

of the Lie algebra of the fifteen operators derived by the Lie method from the two-dimensional damped harmonic oscillator equations $u'' + 2g_1u' + k_1u = 0$ and $v'' + 2g_2v' + k_2v = 0$ ($h = \sqrt{k_j - g_j^2}$, $j = 1, 2$). Their commutators have many combinations of constants, unlike those of previous examples.

On the printout on the following 6 pages are the commutators [L,N] (twice), the anticommutator LN + NL, and the commutator [M,N] and that of the operators Q_1 and Q_9 of the quantum mechanical two-dimensional hydrogen atom, as reported in Anderson et al. (7), pp. 38 to 44 and p. 54:

* The use of the identity $K\Psi = 0$ follows that given in Appendix II, p. 54, Anderson et al. (7). See also the last problem of Example 5, pp. 75 and 76, below.

```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  46 MSEC, COUNT:  74 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
EXP((G1+IH)X)U|(DX-(G1-IH)UDU-(G2-IH)VDV)
#
EXP((-2IH)X)|(DX-(G1+IH)UDU-(G2+IH)VDV)
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (567 MSEC, 826 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  1815 MSEC, 3398 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  1901 MSEC, 3947 STATEMENTS.

 ORDERED RESULT:


 +< 2IHG2EXP((G1+IH)X)UEXP((-2IH)X)V +2IHIHEXP((G1+IH)X)UEXP((-2IH)X)V +G1G2EXP((-2IH)X)EXP((G1+IH)X)UV
 +IHG2EXP((-2IH)X)EXP((G1+IH)X)UV -G1IHEXP((-2IH)X)EXP((G1+IH)X)UV -IHIHEXP((-2IH)X)EXP((G1+IH)X)UV
 -G1G2EXP((-2IH)X)UEXP((G1+IH)X)V -IHG2EXP((-2IH)X)UEXP((G1+IH)X)V +G1IHEXP((-2IH)X)UEXP((G1+IH)X)V
 +IHIHEXP((-2IH)X)UEXP((G1+IH)X)V +G2G2EXP((G1+IH)X)UVEXP((-2IH)X) -IHG2EXP((G1+IH)X)UVEXP((-2IH)X)
 +G2IHEXP((G1+IH)X)UVEXP((-2IH)X) -IHIHEXP((G1+IH)X)UVEXP((-2IH)X) -G2G2EXP((-2IH)X)VEXP((G1+IH)X)U
 -IHG2EXP((-2IH)X)VEXP((G1+IH)X)U +G2IHEXP((-2IH)X)VEXP((G1+IH)X)U +IHIHEXP((-2IH)X)VEXP((G1+IH)X)U >DV

 +< 2IHG1EXP((G1+IH)X)UEXP((-2IH)X)U +2IHIHEXP((G1+IH)X)UEXP((-2IH)X)U +G1G1EXP((-2IH)X)EXP((G1+IH)X)UU
 +IHG1EXP((-2IH)X)EXP((G1+IH)X)UU -G1IHEXP((-2IH)X)EXP((G1+IH)X)UU -IHIHEXP((-2IH)X)EXP((G1+IH)X)UU
 +G1G1EXP((G1+IH)X)UUEXP((-2IH)X) -IHG1EXP((G1+IH)X)UUEXP((-2IH)X) +G1IHEXP((G1+IH)X)UUEXP((-2IH)X)
 -IHIHEXP((G1+IH)X)UUEXP((-2IH)X) -2G1G1EXP((-2IH)X)UEXP((G1+IH)X)U -2IHG1EXP((-2IH)X)UEXP((G1+IH)X)U
 +2G1IHEXP((-2IH)X)UEXP((G1+IH)X)U +2IHIHEXP((-2IH)X)UEXP((G1+IH)X)U >DU

 +< -2IHEXP((G1+IH)X)UEXP((-2IH)X) -G1EXP((-2IH)X)EXP((G1+IH)X)U -IHEXP((-2IH)X)EXP((G1+IH)X)U
 +G1EXP((-2IH)X)UEXP((G1+IH)X) +IHEXP((-2IH)X)UEXP((G1+IH)X) >DX


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  2045 MSEC, 4282 STATEMENTS.
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  2767 MSEC, 5696 STATEMENTS.

 SIMPLIFIED RESULT:


 +< 2IHG2VUEXP((G1+IH)X)EXP((-2IH)X) -2H@(2)VUEXP((G1+IH)X)EXP((-2IH)X) >DV

 +< 2IG1HU@(2)EXP((G1+IH)X)EXP((-2IH)X) -2H@(2)U@(2)EXP((G1+IH)X)EXP((-2IH)X) >DU

 +< -2IHUEXP((G1+IH)X)EXP((-2IH)X) >DX


* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  2809 MSEC, COUNT:  5773 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
EXP((G1+IH)X)U|(DX-(G1-IH)UDU-(G2-IH)VDV)
#
EXP((-2IH)X)|(DX-(G1+IH)UDU-(G2+IH)VDV)
:
```

[L,N]

```
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (3447 MSEC, 6512 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  4204 MSEC, 8149 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  4293 MSEC, 8698 STATEMENTS.

 ORDERED RESULT:


 +< 2IHG2EXP((G1+IH)X)UEXP((-2IH)X)V +2IHIHEXP((G1+IH)X)UEXP((-2IH)X)V +G1G2EXP((-2IH)X)EXP((G1+IH)X)UV
 +IHG2EXP((-2IH)X)EXP((G1+IH)X)UV -G1IHEXP((-2IH)X)EXP((G1+IH)X)UV -IHIHEXP((-2IH)X)EXP((G1+IH)X)UV
 -G1G2EXP((-2IH)X)UEXP((G1+IH)X)V -IHG2EXP((-2IH)X)UEXP((G1+IH)X)V +G1IHEXP((-2IH)X)UEXP((G1+IH)X)V
 +IHIHEXP((-2IH)X)UEXP((G1+IH)X)V +G2G2EXP((G1+IH)X)UVEXP((-2IH)X) -IHG2EXP((G1+IH)X)UVEXP((-2IH)X)
 +G2IHEXP((G1+IH)X)UVEXP((-2IH)X) -IHIHEXP((G1+IH)X)UVEXP((-2IH)X) -G2G2EXP((-2IH)X)VEXP((G1+IH)X)U
 -IHG2EXP((-2IH)X)VEXP((G1+IH)X)U +G2IHEXP((-2IH)X)VEXP((G1+IH)X)U +IHIHEXP((-2IH)X)VEXP((G1+IH)X)U >DV

 +< 2IHG1EXP((G1+IH)X)UEXP((-2IH)X)U +2IHIHEXP((G1+IH)X)UEXP((-2IH)X)U +G1G1EXP((-2IH)X)EXP((G1+IH)X)UU
 +IHG1EXP((-2IH)X)EXP((G1+IH)X)UU -G1IHEXP((-2IH)X)EXP((G1+IH)X)UU -IHIHEXP((-2IH)X)EXP((G1+IH)X)UU
 +G1G1EXP((G1+IH)X)UUEXP((-2IH)X) -IHG1EXP((G1+IH)X)UUEXP((-2IH)X) +G1IHEXP((G1+IH)X)UUEXP((-2IH)X)
 -IHIHEXP((G1+IH)X)UUEXP((-2IH)X) -2G1G1EXP((-2IH)X)UEXP((G1+IH)X)U -2IHG1EXP((-2IH)X)UEXP((G1+IH)X)U
 +2G1IHEXP((-2IH)X)UEXP((G1+IH)X)U +2IHIHEXP((-2IH)X)UEXP((G1+IH)X)U >DU

 +< -2IHEXP((G1+IH)X)UEXP((-2IH)X) -G1EXP((-2IH)X)EXP((G1+IH)X)U -IHEXP((-2IH)X)EXP((G1+IH)X)U
 +G1EXP((-2IH)X)UEXP((G1+IH)X) +IHEXP((-2IH)X)UEXP((G1+IH)X) >DX


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  4439 MSEC, 9033 STATEMENTS.
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  4771 MSEC, 9930 STATEMENTS.

 SIMPLIFIED RESULT:

 +< 2IHG2VUEXP((G1+IH)X)EXP((-2IH)X) -2H@(2)VUEXP((G1+IH)X)EXP((-2IH)X) >DV

 +< 2IG1HU@(2)EXP((G1+IH)X)EXP((-2IH)X) -2H@(2)U@(2)EXP((G1+IH)X)EXP((-2IH)X) >DU

 +< -2IHUEXP((G1+IH)X)EXP((-2IH)X) >DX



* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  4813 MSEC, COUNT:  10007 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
EXP((G1+IH)X)U|(DX-(G1-IH)UDU-(G2-IH)VDV)
#
EXP((-2IH)X)|(DX-(G1+IH)UDU-(G2+IH)VDV)
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (5447 MSEC, 10746 STATEMENTS)
ANTI
 TIME AND COUNT AT END OF ANTICOMMUTATION AND BEGINNING OF SIMPLIFICATION (SIMCO):  5742 MSEC, 12038 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  6002 MSEC, 13482 STATEMENTS.

 ORDERED RESULT:
```

[L,N]

```
+< G2G2EXP((G1+IH)X)UVEXP((-2IH)X)V -IHG2EXP((G1+IH)X)UVEXP((-2IH)X)V +G2IHEXP((G1+IH)X)UVEXP((-2IH)X)V
-IHIHEXP((G1+IH)X)UVEXP((-2IH)X)V +G2G2EXP((-2IH)X)VEXP((G1+IH)X)UV +IHG2EXP((-2IH)X)VEXP((G1+IH)X)UV
-G2IHEXP((-2IH)X)VEXP((G1+IH)X)UV -IHIHEXP((-2IH)X)VEXP((G1+IH)X)UV >DVDV

+< G1G2EXP((G1+IH)X)UUEXP((-2IH)X)V -IHG2EXP((G1+IH)X)UUEXP((-2IH)X)V +G1IHEXP((G1+IH)X)UUEXP((-2IH)X)V
-IHIHEXP((G1+IH)X)UUEXP((-2IH)X)V +G1G2EXP((-2IH)X)UEXP((G1+IH)X)UV +IHG2EXP((-2IH)X)UEXP((G1+IH)X)UV
-G1IHEXP((-2IH)X)UEXP((G1+IH)X)UV -IHIHEXP((-2IH)X)UEXP((G1+IH)X)UV +G2G1EXP((G1+IH)X)UVEXP((-2IH)X)U
-IHG1EXP((G1+IH)X)UVEXP((-2IH)X)U +G2IHEXP((G1+IH)X)UVEXP((-2IH)X)U -IHIHEXP((G1+IH)X)UVEXP((-2IH)X)U
+G2G1EXP((-2IH)X)VEXP((G1+IH)X)UU +IHG1EXP((-2IH)X)VEXP((G1+IH)X)UU -G2IHEXP((-2IH)X)VEXP((G1+IH)X)UU
-IHIHEXP((-2IH)X)VEXP((G1+IH)X)UU >DVDU

+< -G2EXP((G1+IH)X)UEXP((-2IH)X)V -IHEXP((G1+IH)X)UEXP((-2IH)X)V -G2EXP((-2IH)X)EXP((G1+IH)X)UV
+IHEXP((-2IH)X)EXP((G1+IH)X)UV -G2EXP((G1+IH)X)UVEXP((-2IH)X) +IHEXP((G1+IH)X)UVEXP((-2IH)X)
-G2EXP((-2IH)X)VEXP((G1+IH)X)U -IHEXP((-2IH)X)VEXP((G1+IH)X)U >DVDX

+< G1G1EXP((G1+IH)X)UUEXP((-2IH)X)U -IHG1EXP((G1+IH)X)UUEXP((-2IH)X)U +G1IHEXP((G1+IH)X)UUEXP((-2IH)X)U
-IHIHEXP((G1+IH)X)UUEXP((-2IH)X)U +G1G1EXP((-2IH)X)UEXP((G1+IH)X)UU +IHG1EXP((-2IH)X)UEXP((G1+IH)X)UU
-G1IHEXP((-2IH)X)UEXP((G1+IH)X)UU -IHIHEXP((-2IH)X)UEXP((G1+IH)X)UU >DUDU

+< -G1EXP((G1+IH)X)UEXP((-2IH)X)U -IHEXP((G1+IH)X)UEXP((-2IH)X)U -G1EXP((-2IH)X)EXP((G1+IH)X)UU
+IHEXP((-2IH)X)EXP((G1+IH)X)UU -G1EXP((G1+IH)X)UUEXP((-2IH)X) +IHEXP((G1+IH)X)UUEXP((-2IH)X)
-G1EXP((-2IH)X)UEXP((G1+IH)X)U -IHEXP((-2IH)X)UEXP((G1+IH)X)U >DUDX

+< EXP((G1+IH)X)UEXP((-2IH)X) +EXP((-2IH)X)EXP((G1+IH)X)U >DXDX


+< 2IHG2EXP((G1+IH)X)UEXP((-2IH)X)V +2IHIHEXP((G1+IH)X)UEXP((-2IH)X)V -G1G2EXP((-2IH)X)EXP((G1+IH)X)UV
-IHG2EXP((-2IH)X)EXP((G1+IH)X)UV +G1IHEXP((-2IH)X)EXP((G1+IH)X)UV +IHIHEXP((-2IH)X)EXP((G1+IH)X)UV
+G1G2EXP((-2IH)X)UEXP((G1+IH)X)V +IHG2EXP((-2IH)X)UEXP((G1+IH)X)V -G1IHEXP((-2IH)X)UEXP((G1+IH)X)V
-IHIHEXP((-2IH)X)UEXP((G1+IH)X)V +G2G2EXP((G1+IH)X)UVEXP((-2IH)X) -IHG2EXP((G1+IH)X)UVEXP((-2IH)X)
+G2IHEXP((G1+IH)X)UVEXP((-2IH)X) -IHIHEXP((G1+IH)X)UVEXP((-2IH)X) +G2G2EXP((-2IH)X)VEXP((G1+IH)X)U
+IHG2EXP((-2IH)X)VEXP((G1+IH)X)U -G2IHEXP((-2IH)X)VEXP((G1+IH)X)U -IHIHEXP((-2IH)X)VEXP((G1+IH)X)U >DV

+< 2IHG1EXP((G1+IH)X)UEXP((-2IH)X)U +2IHIHEXP((G1+IH)X)UEXP((-2IH)X)U -G1G1EXP((-2IH)X)EXP((G1+IH)X)UU
-IHG1EXP((-2IH)X)EXP((G1+IH)X)UU +G1IHEXP((-2IH)X)EXP((G1+IH)X)UU +IHIHEXP((-2IH)X)EXP((G1+IH)X)UU
+G1G1EXP((G1+IH)X)UUEXP((-2IH)X) -IHG1EXP((G1+IH)X)UUEXP((-2IH)X) +G1IHEXP((G1+IH)X)UUEXP((-2IH)X)
-IHIHEXP((G1+IH)X)UUEXP((-2IH)X) +2G1G1EXP((-2IH)X)UEXP((G1+IH)X)U +2IHG1EXP((-2IH)X)UEXP((G1+IH)X)U
-2G1IHEXP((-2IH)X)UEXP((G1+IH)X)U -2IHIHEXP((-2IH)X)UEXP((G1+IH)X)U >DU

+< -2IHEXP((G1+IH)X)UEXP((-2IH)X) +G1EXP((-2IH)X)EXP((G1+IH)X)U +IHEXP((-2IH)X)EXP((G1+IH)X)U
-G1EXP((-2IH)X)UEXP((G1+IH)X) -IHEXP((-2IH)X)UEXP((G1+IH)X) >DX


TIME AND COUNT AT END OF OUTPUT ROUTINE:  6474 MSEC, 14284 STATEMENTS.
TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  7367 MSEC, 16508 STATEMENTS.

SIMPLIFIED RESULT:


+< 2G2@(2)V@(2)UEXP((G1+IH)X)EXP((-2IH)X) +2H@(2)V@(2)UEXP((G1+IH)X)EXP((-2IH)X) >DVDV

+< 4G1G2VU@(2)EXP((G1+IH)X)EXP((-2IH)X) +4H@(2)VU@(2)EXP((G1+IH)X)EXP((-2IH)X) >DVDU

+< -4G2VUEXP((G1+IH)X)EXP((-2IH)X) >DVDX

+< 2G1@(2)U@(3)EXP((G1+IH)X)EXP((-2IH)X) +2H@(2)U@(3)EXP((G1+IH)X)EXP((-2IH)X) >DUDU

+< -4G1U@(2)EXP((G1+IH)X)EXP((-2IH)X) >DUDX
```

LN+NL

```
+< 2UEXP((G1+IH)X)EXP((-2IH)X) >DXDX

+< 2IHG2VUEXP((G1+IH)X)EXP((-2IH)X) +2G2@(2)VUEXP((G1+IH)X)EXP((-2IH)X) >DV

+< 2IG1HU@(2)EXP((G1+IH)X)EXP((-2IH)X) +2G1@(2)U@(2)EXP((G1+IH)X)EXP((-2IH)X) >DU

+< -2IHUEXP((G1+IH)X)EXP((-2IH)X) >DX



* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  7460 MSEC, COUNT:  16705 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
EXP((G1-IH)X)U|(DX-(G1+IH)UDU-(G2+IH)VDV)
#
EXP((-2IH)X)|(DX-(G1+IH)UDU-(G2+IH)VDV)
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (8118 MSEC, 17453 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  9076 MSEC, 19366 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  9164 MSEC, 19915 STATEMENTS.

 ORDERED RESULT:


 +< 2IHG2EXP((G1-IH)X)UEXP((-2IH)X)V +2IHIHEXP((G1-IH)X)UEXP((-2IH)X)V +G1G2EXP((-2IH)X)EXP((G1-IH)X)UV
 -IHG2EXP((-2IH)X)EXP((G1-IH)X)UV +G1IHEXP((-2IH)X)EXP((G1-IH)X)UV -IHIHEXP((-2IH)X)EXP((G1-IH)X)UV
 -G1G2EXP((-2IH)X)UEXP((G1-IH)X)V -IHG2EXP((-2IH)X)UEXP((G1-IH)X)V -G1IHEXP((-2IH)X)UEXP((G1-IH)X)V
 -IHIHEXP((-2IH)X)UEXP((G1-IH)X)V +G2G2EXP((G1-IH)X)UVEXP((-2IH)X) +IHG2EXP((G1-IH)X)UVEXP((-2IH)X)
 +G2IHEXP((G1-IH)X)UVEXP((-2IH)X) +IHIHEXP((G1-IH)X)UVEXP((-2IH)X) -G2G2EXP((-2IH)X)VEXP((G1-IH)X)U
 -IHG2EXP((-2IH)X)VEXP((G1-IH)X)U -G2IHEXP((-2IH)X)VEXP((G1-IH)X)U -IHIHEXP((-2IH)X)VEXP((G1-IH)X)U >DV

 +< 2IHG1EXP((G1-IH)X)UEXP((-2IH)X)U +2IHIHEXP((G1-IH)X)UEXP((-2IH)X)U +G1G1EXP((-2IH)X)EXP((G1-IH)X)UU
 -IHG1EXP((-2IH)X)EXP((G1-IH)X)UU +G1IHEXP((-2IH)X)EXP((G1-IH)X)UU -IHIHEXP((-2IH)X)EXP((G1-IH)X)UU
 +G1G1EXP((G1-IH)X)UUEXP((-2IH)X) +IHG1EXP((G1-IH)X)UUEXP((-2IH)X) +G1IHEXP((G1-IH)X)UUEXP((-2IH)X)
 +IHIHEXP((G1-IH)X)UUEXP((-2IH)X) -2G1G1EXP((-2IH)X)UEXP((G1-IH)X)U -2IHG1EXP((-2IH)X)UEXP((G1-IH)X)U
 -2G1IHEXP((-2IH)X)UEXP((G1-IH)X)U -2IHIHEXP((-2IH)X)UEXP((G1-IH)X)U >DU

 +< -2IHEXP((G1-IH)X)UEXP((-2IH)X) -G1EXP((-2IH)X)EXP((G1-IH)X)U +IHEXP((-2IH)X)EXP((G1-IH)X)U
 +G1EXP((-2IH)X)UEXP((G1-IH)X) +IHEXP((-2IH)X)UEXP((G1-IH)X) >DX


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  9307 MSEC, 20250 STATEMENTS.

 NULL RESULT.

* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  9853 MSEC, COUNT:  21471 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
EXP(IV)|(IR@(-1)DVDV+DRDV+.5IDR-.5R@(-1)DV+.5DT)
```

[M,N]

```
#
EXP(IT-IV)|(-R@(-1)DVDV +DRDV -(1+R)DR -.5(R@(-1)+1)DV -.5DT +.25IR)
:
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (10730 MSEC, 22396 STATEMENTS)
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  14840 MSEC, 28764 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  15340 MSEC, 31535 STATEMENTS.

 ORDERED RESULT:


 +< -IEXP(IT-IV)EXP(IV) -IEXP(IV)EXP(IT-IV) >DRDRDV

 +< -IIEXP(IV)R@(-1)EXP(IT-IV) -IIEXP(IV)R@(-1)EXP(IT-IV) +IEXP(IT-IV)R@(-1)EXP(IV) +IEXP(IT-IV)R@(-1)EXP(IV)
 -IIEXP(IT-IV)EXP(IV)R@(-1) +IEXP(IV)EXP(IT-IV)R@(-1) >DRDVDV

 +< IIEXP(IV)R@(-1)EXP(IT-IV)R@(-1) +IIEXP(IV)R@(-1)EXP(IT-IV)R@(-1) +IIEXP(IT-IV)R@(-1)EXP(IV)R@(-1)
 +IIEXP(IT-IV)R@(-1)EXP(IV)R@(-1) +IEXP(IT-IV)EXP(IV)R@(-2) +EXP(IV)EXP(IT-IV)R@(-2) >DVDVDV


 +< -0.5IEXP(IT-IV)EXP(IV) +0.5IEXP(IV)EXP(IT-IV) >DTDR

 +< 0.5IIEXP(IV)R@(-1)EXP(IT-IV) +0.5IIEXP(IV)R@(-1)EXP(IT-IV) +0.5IEXP(IT-IV)R@(-1)EXP(IV)
 +0.5IEXP(IT-IV)R@(-1)EXP(IV) >DTDV

 +< -0.5IIEXP(IT-IV)EXP(IV) +IEXP(IV)EXP(IT-IV) +IEXP(IV)EXP(IT-IV)R >DRDR

 +< IIIEXP(IV)R@(-1)EXP(IT-IV) +IIEXP(IT-IV)R@(-1)EXP(IV) +IIEXP(IV)R@(-1)EXP(IT-IV)
 +IIEXP(IV)R@(-1)EXP(IT-IV)R +IIEXP(IV)R@(-1)EXP(IT-IV) +IIEXP(IV)R@(-1)EXP(IT-IV)R
 +0.5IIEXP(IT-IV)R@(-1)EXP(IV) +0.5IIEXP(IT-IV)R@(-1)EXP(IV) -EXP(IV)EXP(IT-IV) +0.5IEXP(IT-IV)EXP(IV)R@(-1)
 +0.5IEXP(IV)EXP(IT-IV)R@(-1) +0.5IEXP(IV)EXP(IT-IV) +0.5IEXP(IT-IV)R@(-1)EXP(IV) +0.5IEXP(IT-IV)EXP(IV)
 +0.5IEXP(IV)R@(-1)EXP(IT-IV) +0.5IEXP(IV)EXP(IT-IV) >DRDV

 +< -IIIEXP(IV)R@(-1)EXP(IT-IV)R@(-1) +IIIEXP(IT-IV)R@(-1)EXP(IV)R@(-1) +0.5IIEXP(IV)R@(-1)EXP(IT-IV)R@(-1)
 +0.5IIEXP(IV)R@(-1)EXP(IT-IV) +0.5IIEXP(IV)R@(-1)EXP(IT-IV)R@(-1) +0.5IIEXP(IV)R@(-1)EXP(IT-IV)
 -0.5IEXP(IT-IV)R@(-1)EXP(IV)R@(-1) -0.5IEXP(IT-IV)R@(-1)EXP(IV)R@(-1) +IIEXP(IT-IV)EXP(IV)R@(-2)
 -IEXP(IV)EXP(IT-IV)R@(-2) -0.5EXP(IT-IV)EXP(IV)R@(-2) +0.5EXP(IV)EXP(IT-IV)R@(-2) -IEXP(IT-IV)EXP(IV)R@(-2)
 -IEXP(IT-IV)REXP(IV)R@(-2) +0.5IEXP(IV)EXP(IT-IV)R@(-2) +0.5IIEXP(IT-IV)R@(-1)EXP(IV)R@(-1)
 +0.5IIEXP(IT-IV)EXP(IV)R@(-1) -0.5IEXP(IV)R@(-1)EXP(IT-IV)R@(-1) -0.5IEXP(IV)EXP(IT-IV)R@(-1) >DVDV


 +< -0.5IIIEXP(IV)R@(-1)EXP(IT-IV) +0.5IIEXP(IT-IV)R@(-1)EXP(IV) +0.25IEXP(IT-IV)R@(-1)EXP(IV)
 +0.25IEXP(IT-IV)EXP(IV) -0.25IEXP(IV)R@(-1)EXP(IT-IV) -0.25IEXP(IV)EXP(IT-IV) >DT

 +< -IIIEXP(IV)R@(-1)EXP(IT-IV) -IIIEXP(IV)R@(-1)EXP(IT-IV)R +0.5IIIEXP(IT-IV)R@(-1)EXP(IV)
 +IEXP(IV)EXP(IT-IV) -0.25IIEXP(IV)EXP(IT-IV)R -0.5IEXP(IV)EXP(IT-IV) +0.25IIEXP(IT-IV)R@(-1)EXP(IV)
 +0.25IIEXP(IT-IV)EXP(IV) -0.5IEXP(IV)R@(-1)EXP(IT-IV) -0.5IEXP(IV)R@(-1)EXP(IT-IV)R -0.5IEXP(IV)EXP(IT-IV)
 -0.5IEXP(IV)EXP(IT-IV)R >DR

 +< -0.5IIIEXP(IV)R@(-1)EXP(IT-IV)R@(-1) -0.5IIIEXP(IV)R@(-1)EXP(IT-IV) -0.5IIEXP(IT-IV)R@(-1)EXP(IV)R@(-1)
 -0.25IIIEXP(IV)R@(-1)EXP(IT-IV)R -0.25IIIEXP(IV)R@(-1)EXP(IT-IV)R -0.5IEXP(IT-IV)EXP(IV)R@(-2)
 -0.5IEXP(IV)EXP(IT-IV)R@(-2) +0.25IEXP(IV)EXP(IT-IV) +0.5EXP(IT-IV)EXP(IV)R@(-2)
 +0.5EXP(IT-IV)REXP(IV)R@(-2) +0.25IEXP(IV)EXP(IT-IV)R@(-2) -0.25IEXP(IT-IV)R@(-1)EXP(IV)R@(-1)
 -0.25IEXP(IT-IV)EXP(IV)R@(-1) -0.25IEXP(IV)R@(-1)EXP(IT-IV)R@(-1) -0.25IEXP(IV)R@(-1)EXP(IT-IV)
 -0.25IEXP(IV)EXP(IT-IV)R@(-1) -0.25IEXP(IV)EXP(IT-IV) >DV


 +< 0.25IIIIEXP(IV)R@(-1)EXP(IT-IV)R -0.25IIEXP(IV)EXP(IT-IV) +0.125IIEXP(IV)EXP(IT-IV)
 +0.125IIEXP(IV)R@(-1)EXP(IT-IV)R +0.125IIEXP(IV)EXP(IT-IV)R >
```

```
TIME AND COUNT AT END OF OUTPUT ROUTINE:  16149 MSEC, 32453 STATEMENTS.
TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  17788 MSEC, 35335 STATEMENTS.

SIMPLIFIED RESULT:


+< -2IEXP(IV)EXP(IT-IV) >DRDRDV

+< 3R@(-1)EXP(IV)EXP(IT-IV) +3IR@(-1)EXP(IV)EXP(IT-IV) >DRDVDV

+< -3R@(-2)EXP(IV)EXP(IT-IV) +IR@(-2)EXP(IV)EXP(IT-IV) >DVDVDV


+< -R@(-1)EXP(IV)EXP(IT-IV) +IR@(-1)EXP(IV)EXP(IT-IV) >DTDV

+< 0.5EXP(IV)EXP(IT-IV) +IEXP(IV)EXP(IT-IV) +IREXP(IV)EXP(IT-IV) >DRDR

+< IR@(-1)EXP(IV)EXP(IT-IV) -4R@(-1)EXP(IV)EXP(IT-IV) +1.5IEXP(IV)EXP(IT-IV) -3EXP(IV)EXP(IT-IV) >DRDV

+< -3IR@(-2)EXP(IV)EXP(IT-IV) -1.5IR@(-1)EXP(IV)EXP(IT-IV) -2.5R@(-2)EXP(IV)EXP(IT-IV)
-1.5R@(-1)EXP(IV)EXP(IT-IV) >DVDV


+< 0.5IR@(-1)EXP(IV)EXP(IT-IV) -0.5R@(-1)EXP(IV)EXP(IT-IV) >DT

+< 0.5IEXP(IV)EXP(IT-IV) -0.25EXP(IV)EXP(IT-IV) -0.5IREXP(IV)EXP(IT-IV) -0.25R@(-1)EXP(IV)EXP(IT-IV)
+0.25REXP(IV)EXP(IT-IV) >DR

+< -0.75IR@(-2)EXP(IV)EXP(IT-IV) -0.25IR@(-1)EXP(IV)EXP(IT-IV) +R@(-2)EXP(IV)EXP(IT-IV)
+0.5IEXP(IV)EXP(IT-IV) +0.5R@(-1)EXP(IV)EXP(IT-IV) >DV


+< 0.25EXP(IV)EXP(IT-IV) -0.125REXP(IV)EXP(IT-IV) >


* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  18140 MSEC, COUNT:  35722 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
TERM
```

$[Q_1, Q_9]$

The compilation and termination statistics are as follows:

```
COMPILATION TIME  4.561  SECONDS
MEMORY USAGE (DECIMAL BYTES)
CODE:              92594
STRINGS:            4831
VARIABLES:         14880
CONSTANTS:          3416
TOTAL:            115721
AVAILABLE:         91543

NORMAL TERMINATION IN STATEMENT     1042
NUMBER OF STATEMENTS EXECUTED      35727
EXECUTION TIME (SECONDS)          18.149
MSEC/STATEMENT                     0.507
NUMBER OF STORAGE REGENERATIONS       45
```

Notice that the first two operators and their commutators are the same. These show that some time is gained by the shortcuts in the derivative and simplification sequences, by which derivative and cannonical ordering tables are constructed to speed processing for repetitious actions. On the printout the times to compare are those for commutation and those for simplification. For the first processing of [L,N] the respective times are 1248 msec (1815 minus 567) and 722 msec (2767 minus 2045); for the second, 757 msec and 332 msec.

The anticommutator of L and N is included to show the use of the "ANTI" instruction.

The commutator [M,N] = 0 is verified in the fourth problem. Notice how well the complexity of the result is reduced in this and in the previous problems of this example.

The final problem in this example, the commutator $[Q_1,Q_9]$ of two operators of the two-dimensional hydrogen atom, was used in Appendix II, Anderson et al. (7), p. 54, to illustrate the reduction of a

result due to an operator identity. There it is shown that the result ($v = \phi$), $A = e^{it}\{-2i\partial_r\partial_r\partial_v - 2i/r^2\partial_v\partial_v\partial_v - 2/r\partial_t\partial_v - \frac{1}{2}r\partial_r\partial_r - 2i/r\partial_r\partial_v - 1/2r\partial_v\partial_v + \frac{1}{2}(r-1)\partial_r + \frac{1}{2}i\partial_v + \frac{1}{4} - r/8\}$ reduces to $-\frac{1}{2}ie^{it}\{ir\partial_r + \partial_t + \frac{1}{2}i - \frac{1}{2}ir\}$ due to the operator identity $\partial_r\partial_r + 1/r\partial_r + 1/r^2\partial_v\partial_v - i/r\partial_t - \frac{1}{4} = 0$.

Example 6. According to a method recently developed by R. L. Anderson, T. Shibuya and C. E. Wulfman*, a constant of the motion in one physical system may be transformed into the same in another physical system by knowing the constant of motion for the first and the hamiltonians for both systems. A simple example is the transformation of the invariant $x + it\partial_x$ from the free particle to the harmonic oscillator system. For a free particle, $x - vt$ is constant. Defining $p \equiv v$ ($m \equiv 1$) and replacing p by $-i\partial_x$, we may consider $Q_a = x + it\partial_x$ to be a constant of motion of the free particle system. The hamiltonians of the free particle and harmonic oscillator systems may be respectively written $H_a = \frac{1}{2}p^2 = -\frac{1}{2}\partial_x\partial_x$ and $H_b = \frac{1}{2}(p^2 + k^2x^2) = -\frac{1}{2}(\partial_x\partial_x - k^2x^2)$.

Now, to transform the invariant Q_a in the free particle system to the invariant Q_b in the harmonic oscillator system, the following identification is made:

$$Q_b = \exp(-iH_bt)\exp(iH_at)Q_a\exp(-iH_at)\exp(iH_bt)$$

* Presented by C. E. Wulfman at the AAAS meeting, Mexico City, Jun 73.

Using the identity $e^{aS}Ye^{-aS} = Y + a[S,Y] + \frac{a^2[S,[S,Y]]}{2!} + \ldots$, the internal "sandwich" $\exp(iH_a t)Q_a\exp(-iH_a t)$ may be shown to be equal to $Q_a + it[H_a,Q_a] + \frac{(it)^2[H_a,[H_a,Q_a]]}{2!} + \ldots = x + it\partial_x - it\partial_x + 0 = x.$

Thus $Q_b = \exp(-iH_b t)x\exp(iH_b t) = x - it[H_b,x] + \frac{(-it)^2[H_b,[H_b,x]]}{2!} + \ldots.$

The terms $-it[H_b,x]$, $(-it)^2[H_b,[H_b,x]]$, ..., to order 8 are evaluated by MANDO when the following set of inputs is used:

```
"-IT|(-.5DXDX+.5K@(2)X@(2))#X:"
"SAVE"
"COMM"
"COMM"
"COMM"
"COMM"
"COMM"
"COMM"
"COMM"
"OPER"
"TERM"
"TERM"
```

The printout follows on the next 4 pages. The compilation and termination statistics are:

```
COMPILATION TIME  4.547  SECONDS
MEMORY USAGE (DECIMAL BYTES)
CODE:              92594
STRINGS:            4831
VARIABLES:         14880
CONSTANTS:          3416
TOTAL:            115721
AVAILABLE:         91543

NORMAL TERMINATION IN STATEMENT    1042
NUMBER OF STATEMENTS EXECUTED      2761
EXECUTION TIME (SECONDS)          1.121
MSEC/STATEMENT                    0.406
NUMBER OF STORAGE REGENERATIONS       2
```

```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  44 MSEC, COUNT:  74 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
-IT|(-.5DXDX +.5K@(2)X@(2)) # X :
 "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR "APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR "CHEK":
 (TIME,COUNT):  (183 MSEC, 270 STATEMENTS)
SAVE

* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* ENTRY INTO EXECUTION CYCLE *$$*
 "COMM", "ANTI", "PROD" OR "APLY":
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  218 MSEC, 398 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  229 MSEC, 480 STATEMENTS.

 ORDERED RESULT:

 +< 0.5IT +0.5IT >DX

                                                                                                        ORDER
                                                                                                        OF TERM*
 TIME AND COUNT AT END OF OUTPUT ROUTINE:  247 MSEC, 516 STATEMENTS.
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  270 MSEC, 583 STATEMENTS.

 SIMPLIFIED RESULT:

 +< IT >DX                                                                                  -it[H_b, x]      2

* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* CONTINUATION OF EXECUTION CYCLE, TIME:  288 MSEC, COUNT:  612 STATEMENTS *$$*
 "COMM", "ANTI", "PROD", "APLY" OR "OPER":
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  332 MSEC, 766 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  338 MSEC, 814 STATEMENTS.

 ORDERED RESULT:

 +< IIK@(2)TTX >

 TIME AND COUNT AT END OF OUTPUT ROUTINE:  351 MSEC, 834 STATEMENTS.
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  384 MSEC, 916 STATEMENTS.

 SIMPLIFIED RESULT:

 +< -K@(2)XT@(2) >                                                                  (-it)^2 [H_b, [H_b, x]]      3
```

*see Table 1.1, p. 3.

```
* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* CONTINUATION OF EXECUTION CYCLE, TIME:  403 MSEC, COUNT:  948 STATEMENTS *$$*
 "COMM", "ANTI", "PROD", "APLY" OR "OPER":
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  440 MSEC, 1094 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  451 MSEC, 1176 STATEMENTS.

 ORDERED RESULT:


 +< -0.5IK@(2)TT@(2) -0.5IK@(2)TT@(2) >DX


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  471 MSEC, 1212 STATEMENTS.
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  549 MSEC, 1290 STATEMENTS.

 SIMPLIFIED RESULT:


 +< -IK@(2)T@(3) >DX                                                    (-it)^3 [H_b,[H_b[H_b,x]]]   4


* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* CONTINUATION OF EXECUTION CYCLE, TIME:  568 MSEC, COUNT:  1327 STATEMENTS *$$*
 "COMM", "ANTI", "PROD", "APLY" OR "OPER":
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  603 MSEC, 1457 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  609 MSEC, 1505 STATEMENTS.

 ORDERED RESULT:


 +< -IK@(2)IK@(2)T@(3)TX >


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  622 MSEC, 1525 STATEMENTS.
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  664 MSEC, 1615 STATEMENTS.

 SIMPLIFIED RESULT:


 +< K@(4)XT@(4) >                                                       (-it)^4 [H_b,[H_b,[H_b,[H_b,x]]]]   5


* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* CONTINUATION OF EXECUTION CYCLE, TIME:  683 MSEC, COUNT:  1647 STATEMENTS *$$*
 "COMM", "ANTI", "PROD", "APLY" OR "OPER":
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  720 MSEC, 1789 STATEMENTS.
```

```
TIME AND COUNT AT END OF ORDERING OF RESULT:  731 MSEC, 1871 STATEMENTS.

ORDERED RESULT:

+< 0.5IK@(4)TT@(4) +0.5IK@(4)TT@(4) >DX


TIME AND COUNT AT END OF OUTPUT ROUTINE:  749 MSEC, 1907 STATEMENTS.
TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  788 MSEC, 2000 STATEMENTS.

SIMPLIFIED RESULT:

+< IK@(4)T@(5) >DX

* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* CONTINUATION OF EXECUTION CYCLE, TIME:  807 MSEC, COUNT:  2037 STATEMENTS *$$*
 "COMM", "ANTI", "PROD", "APLY" OR "OPER":
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  840 MSEC, 2164 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  846 MSEC, 2212 STATEMENTS.

 ORDERED RESULT:

 +< IK@(4)IK@(2)T@(5)TX >


 TIME AND COUNT AT END OF OUTPUT ROUTINE:  859 MSEC, 2232 STATEMENTS.
 TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  901 MSEC, 2321 STATEMENTS.

 SIMPLIFIED RESULT:

 +< -K@(6)XT@(6) >

* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* CONTINUATION OF EXECUTION CYCLE, TIME:  921 MSEC, COUNT:  2353 STATEMENTS *$$*
 "COMM", "ANTI", "PROD", "APLY" OR "OPER":
COMM
 TIME AND COUNT AT END OF COMMUTATION AND BEGINNING OF SIMPLIFICATION FUNCTION (SIMCO):  958 MSEC, 2499 STATEMENTS.
 TIME AND COUNT AT END OF ORDERING OF RESULT:  969 MSEC, 2581 STATEMENTS.

 ORDERED RESULT:

 +< -0.5IK@(6)TT@(6) -0.5IK@(6)TT@(6) >DX
```

6

7

```
TIME AND COUNT AT END OF OUTPUT ROUTINE:  987 MSEC, 2617 STATEMENTS.
TIME AND COUNT AT END OF SIMPLIFICATION OF RESULT:  1081 MSEC, 2709 STATEMENTS.

SIMPLIFIED RESULT:

+< -IK@(6)T@(7) >DX

* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* CONTINUATION OF EXECUTION CYCLE, TIME:  1100 MSEC, COUNT:  2746 STATEMENTS *$$*
 "COMM", "ANTI", "PROD", "APLY" OR "OPER":
OPER

* -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- ** -- *

*$$* BEGINNING OF INPUT SEQUENCE, TIME:  1111 MSEC, COUNT:  2756 STATEMENTS *$$*
 OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:
TERM
```

8

It is evident from the printout that $Q_b = x + it_x - \frac{k^2t^2x}{2!}$

$- \frac{ik^2t^3}{3!}\partial_x + \frac{k^4t^4x}{4!} + \frac{ik^4t^5}{5!}\partial_x - \frac{k^6t^6x}{6!} - \frac{ik^6t^7}{7!}\partial_x + \ldots = x(1 - \frac{k^2t^2}{2!}$

$+ \frac{k^4t^4}{4!} - \frac{k^6t^6}{6!} + \ldots) + i/k(kt - \frac{k^3t^3}{3!} + \frac{k^5t^5}{5!} - \frac{k^7t^7}{7!} + \ldots)\partial_x$

$= x\cdot\cos(kt) + i/k\cdot\sin(kt)\partial_x = x\cdot\cos(kt) - p/k\cdot\sin(kt)$. This result is equal to a linear combination of invariants of the harmonic oscillator system*.

* $1/2k\cdot(Q_2 - Q_3)$ for the Q_2 and Q_3 in the column "$v = \frac{1}{2}kx^2$" of Table I, Anderson et al. (7), p. 16, and k replaced by k^2. The operator Q_a in this example is the Q_2 in column "v = 0" of Table I.

Chapter V

Technical Description of Program

To understand this chapter, the reader will be aided by a working knowledge of the SNOBOL programming language (5). Frequent references are made to the program listing, which is Appendix B. Line numbers given here refer to the ones in that listing. Since there may be confusion in using the word "function" to represent both mathematical expressions and discrete units within the program (as in SNOBOL), the latter are called "p-functions" (for program functions); similarly, program variables are called "p-variables".

5.1 Internal Data Structure; Datatypes OPDER, EXFUN and ADDEX

The program MANDO stores differential operators in nested arrays and defined datatypes. This facilitates the referencing of the relevant additive components of operators, of additive components of functions which multiply derivative coefficients within the operators, and of the elements of the latter. The programmer-defined datatypes in MANDO are labelled OPDER, EXFUN and ADDEX. Figures 5.1.1 and 5.1.2, pp. 88 and 89, summarize the discussion of this section and may be helpful in learning about these datatypes.

The fields within OPDER are labelled OI and OA. The field OA contains a rectangular array. The array may have dimension 2, 3 or 4 in one direction, the number depending on the application; in

the other direction the array has a dimension N equal to the number of additive operator components, that is, terms which may be considered as additive components of the operator as a whole, each associated with a set of derivative coefficients, e. g., the operators $y\partial_x - x\partial_y$ and $x(x\partial_x + y\partial_y)$ each have two such additive operator components. The field OI stores the value of the number N. If OPER is an operator, then OPER = OPDER(OI,OA), where OI(OPER) = N and OA(OPER) = ARRAY(4,N) ("OA(OPER) = ARRAY(4,N)" is not a valid SNOBOL assignment statement but is clearer here than "OA(OPER) = ARRAY('4,' N)", which is valid).

To each additive operator component there is associated a function and a set of derivative coefficients, e. g., in the second component of operator $\exp(-2ihx)(\partial_x - (g_1 + ih)u\partial_u - (g_2 + ih)v\partial_v)$ the function is $-(g_1 + ih)u\cdot\exp(-2ihx)$, and the set of derivative coefficients, ∂_u. The first of the four array elements referenced when the second array index has a fixed value, say, K, OA(OPER)<1,K>, stores the function associated with the additive operator component K.

The other three (or two or one) array elements indexed by K store information about the set of derivative coefficients. OA(OPER)<2,K>, the second element, stores an integer ("lexical ordering number") which represents the set of derivative coefficients. The structure of this integer is of some interest. All variables recognized in the input are tabulated so that they can be uniquely referenced, via table CMP, by other, sequentially ordered integers 1,

2, 3, ...; additionally, powers of 10 are similarly associated with the variables (via table IN). Using these two associations, any set of derivative coefficients up to ninth order can be uniquely obtained from its lexical ordering number and vice versa, e. g., if the variable x is associated with the units and the variable y associated with the 10's, the number 24 represents $\partial_y\partial_y\partial_x\partial_x\partial_x\partial_x$ and 30 represents $\partial_y\partial_y\partial_y$. Use of the lexical ordering numbers permits a rapid evaluation of the result of adding an extra derivative coefficient to an ordered set to produce another ordered set, e. g., the ordered result of adding ∂_y to the ordered set $\partial_x\partial_y\partial_z$ is $\partial_x\partial_y\partial_y\partial_z$. Lexical ordering numbers also control the order in which the additive operator components and in which, among themselves, the derivative coefficients of a set appear on output; thus $\partial_y\partial_y\partial_x$ appears before $\partial_y\partial_x\partial_x$, which appears before $\partial_x\partial_x\partial_x$, where y is associated with the 10's and x, the units (note that the lexical ordering numbers here are 21, 12 and 3, respectively).

The other two array elements referenced by K, OA(OPER)<3,K> and OA(OPER)<4,K>, are not always needed, so they are dropped in certain stages of the program. OA(OPER)<3,K> stores the order of the additive operator component; this value is referenced in p-functions CYHAC (line 463), CYMUL (line 544) and CYAPL (line 595) to terminate their cyclic routines and early in p-function SIMCO (see lines 868 to 871) to classify the additive operator components according to order. OA(OPER)<4,K> stores a string of variables which are those

appearing in the set of derivative coefficients; the variables are associated with p-variables VAR, VR2, VR3, ..., VR9 in p-function OPCOM (lines 436 and 450), OPMUL (lines 520 and 535) and OPAPL (line 586), to be used in the initial calls of CYHAC (lines 440 and 454), CYMUL (lines 524 and 539) and CYAPL (line 596), respectively.

After processing by SIMCO, the structure of the operator is OPER = ARRAY(0:9) (actually ARRAY('0:9') in SNOBOL), and OPER<K>, $0 \leq K \leq C$, where C is the order of the result, is of datatype OPDER and comprises the subset of additive operator components of order K. This structure is required for output processing by p-function OPOUT (lines 1055 to 1109), called within p-function SIMCO (lines 1004 and 1051). When the program is acting under the instruction "SAVE", the structure of the operator may need to be returned to OPDER(N,ARRAY(4,N)); this is done, when required, by p-function RESET (lines 1257 to 1265).

Like OPDER, datatype EXFUN has two fields, EI and EA. The field EA contains a vector array in which are stored the separate additive components of a function associated with an additive operator component, e. g., the second additive operator component of operator $\exp(-2ihx)(\partial_x - (g_1 + ih)u\partial_u - (g_2 + ih)v\partial_v)$ is $-(g_1 + ih)u\cdot\exp(-2ihx)\partial_u$; the additive components of the function associated with this are (expanded, as is done): $-g_1 u\cdot\exp(-2ihx)$ and $-ihu\cdot\exp(-2ihx)$. If FCT is a function whose number of additive components is, say, P, then FCT = EXFUN(EI,EA), where EI(FCT) = P and EA(FCT) = ARRAY(P); as an exception, however, if P = 1, EA(FCT) is usually not an array but is of

datatype ADDEX (explained below)*.

The elements of the array which comprise the EA field of datatype EXFUN are additive components of functions. These components may be further broken down into elements which are multiplicative factors, specifically, the "five elements of additive components" discussed in Chapter II (pp. 6 and 7). If ACO is an additive component, ACO = ADDEX(AS,AN,AC,AT,AD), a datatype with five fields. The values of the first four are strings; AD(ACO) is either null or of datatype EXFUN. Divisors within divisors, which correspond to the AD fields of the additive components of AD(ACO), are not permitted. As before (Chapter II), AS(ACO) can be "+", "-" or null; AN(ACO) is any real number or integer; AC(ACO) is a factor composed of constants; and AT(ACO) is a factor composed of functions of the variables.

The structure of an arbitrary operator is drawn in Figure 5.1.1**. A particular operator is drawn in Figure 5.1.2.

* EA(FCT) = ARRAY(1) is permitted early in p-function SIMCO but is changed back by p-function ADSUB, called near the end of SIMCO.

** See Earley (8) for a discussion of such graphs.

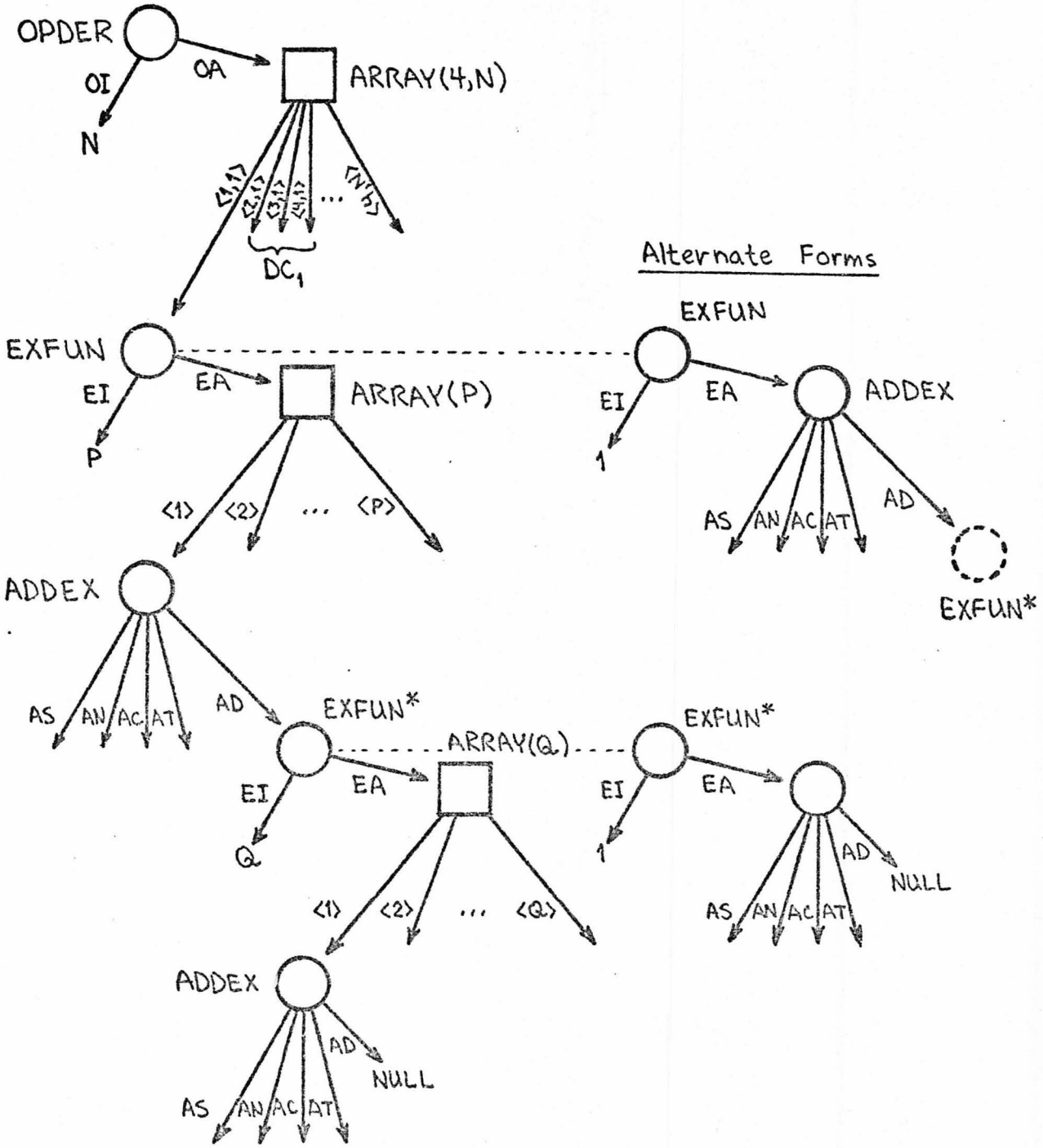

Figure 5.1.1 The structure of an arbitrary operator. An EXFUN structure marked with an asterisk is an AD (divisor) field of an ADDEX structure and is absent when there is no divisor in the additive component (of a function) stored in the ADDEX structure. The alternate forms replace the others connected to them by the dotted lines when the functions they contain cannot be written as sums of additive components (when P or Q equals 1).

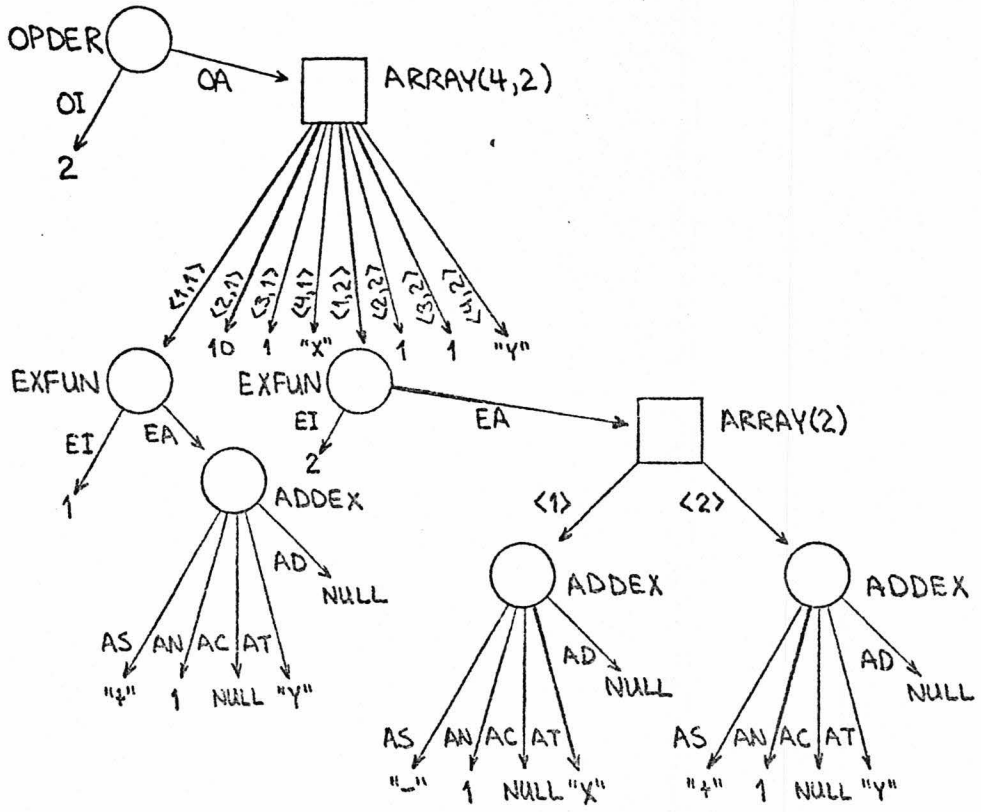

Figure 5.1.2 The structure of operator $y\partial_x - (x - y)\partial_y$.

5.2 Cannonical Ordering and Simplification

As is evident from Examples 4 and 5, Chapter IV, pp. 60 to 76, the difference between the "ordered result" and the "simplified result" in particular problem can be quite remarkable. The differences are due to the program's capability of recognizing that the value of certain expressions is zero and that certain other terms may be added and combined into one. Prerequisite to this capability is the identification and ordering of all variables, constants and functions which may appear in the result.

The p-function SQEZ1 (lines 204 to 220) identifies variables and constants directly from the input and assigns them, in order of

appearance, to elements of the table CMP such as CMP<N>, where N is 1 to 8 for variables and 41 or greater for constants.

P-function SQEZ2 (lines 355 to 391) analyzes the AT fields of components of functions translated to datatype ADDEX and stores the "useful functions" EXP, SIN, COS, LGN and SQT and parenthesized factors in table CMP also. For the "useful functions" the elements CMP<11> to CMP<15> are used, and for parenthesized factors elements CMP<31> to CMP<39> are used. Again, values are assigned to these elements only if the factors appear in the functions, and they are assigned in order of appearance. For a "useful function", the identification of the function is found in CMP<K><1>, where $11 \leq K \leq 15$. If the same function is found in several places with different arguments, the arguments are found in CMP<K><2>, CMP<K><3>, ... Thus, if, for exponential functions, only "EXP(IT)EXP(IV-IT)" may occur as a factor in the result and if CMP<K> corresponds to the exponential functions, CMP<K><1> = "EXP(", CMP<K><2> = "IT", CMP<K><3> = "IV-IT", and CMP<K><0> = 3, the latter, 3, being the highest index storing an argument, a value used in controlling increments. When a sine or cosine function appears, the other corresponding function is also tabulated, as it may appear in the result, the derivative of the sine being the cosine and vice versa (with a sign change).

Since, when parenthesized factors in the functions are not raised to a power, the p-function CONEX expands the components in which the factors appear, such factors are not tabulated as they may never find their way into the results.

CMP<0> is a four element array. CMP<0><1>, CMP<0><2>, CMP<0><3> and CMP<0><4> store, respectively, the highest indices storing variables, "useful functions", parenthesized factors, and constants. They are initialized at 0, 10, 30 and 40, and they control increments.

After the result has been ordered by sets of derivative coefficients, the program proceeds to simplify it. This is begun by changing the factors in the additive components of functions so that factors in separate components are ordered in the same way (according to their order in table CMP but reversed for variables). Thus, for example, "EXP(-IT+IV)XEXP(IT)X" and "EXP(IT)EXP(-IT+IV)XX" may both become "X@(2)EXP(-IT+IV)EXP(IT)". Components with identical such AT fields may be added if their constant and divisor (AC and AD) fields are also the same*. The constant field is ordered in the same manner as the function (AT) field, but the divisor is left alone, as the awkwardness of comparing divisors by referencing each string in them (which must be done unless the divisors are associated in some way, such as having arisen in differentiation - the datatypes are not transparent to the primitive function IDENT in SPITBOL) is overcome by eliminating divisors altogether and writing factor with negative

* This approach to simplification is an incomplete version of a system which appears to aspire to what Moses (9) calls a "radical" system since little choice is left for the output format, except for the order in which the terms may appear, though parenthesized expressions raised to a power, not normally permitted in a "radical" system, are allowed here.

exponents; thus the simplification is more direct and complete if there are no divisors in the functions.

For Example 4, Chapter IV, pp. 60 to 68, divisors in the functions are accordingly expressed as factors with negative exponents. Though the "simplified result" is an improvement over the "ordered result", it still leaves much to be desired in the way of reducing the expression to its simplest form. The improvement of this aspect of the simplification is the subject of Section 6.4, p. 132.

5.3 Program Input and Termination Sequences and Details of P-Function Operation

In the program input sequence a square array OP = ARRAY(2,2) (actually ARRAY('2,2') in SNOBOL) is created in which the pair of operators and their left multipliers are stored in respective elements OP<2,K> and OP<1,K>, K = 1, 2. If the conversion of one of the operators fails, a diagnostic dump occurs which can be examined for the cause of the error. If the first operator is not present, the input is rejected; if the second operator is not present, the first operator is converted before the input is rejected. As was mentioned in Chapter III, p. 18, and will be discussed further on p. 117, this is a means of controlling the ordering of the expressions in the output, since the conversion sets up the variables, constants, "useful functions", and parenthesized factors in a table which is sequentially referenced (reversed for variables) for ordering of the output.

Execution following the instruction "SAVE" is controlled by the segment in lines 119 to 167. The one-time "COMM", "PROD", "ANTI" and "APLY" operations are controlled by the segment in lines 169 to 201.

The instruction "CHEK" results in a special SNOBOL dump of the values of the variables, keywords, and the structures of the operators and of the tables CMP and IN. An example of a dump resulting from the inputs:

```
"EXP((G1+IH)X)U|(DX-(G1-IH)UDU-(G2-IH)VDV)#"
"EXP((-2IH)X)|(DX-(G1+IH)UDU-(G2+IH)VDV):"
"CHEK"
```

appears on the next two pages. The dump resulting from:

```
"-I(1-X@(2))@(-.5)DRDV-.5I(1-X@(2))@(-.5)DV#"
"(X-X@(3))/(R)DXDX:"
"CHEK"
```

appears on the third page following (for brevity several pages of printout are compressed here - note that, in the current version of the program, pattern BPR is not used and &STLIMIT is set at 200,000 instead of at 50,000, the default value):

DUMP OF NATURAL VARIABLES

```
ABORT = PATTERN
ACN = PATTERN
ANY = PATTERN
BAL = PATTERN
BPR = PATTERN
CCF = TABLE(11) #5
CMP = TABLE(11) #3
CNA = PATTERN
CNP = PATTERN
CNS = PATTERN
DTA = TABLE(21) #7
FAD = EXFUN #2
FCF = TABLE(21) #6
FENCE = PATTERN
IN = TABLE(3) #9
INPUT = 'CHEK'
J = 1
K = 29
L = 3
NCN = PATTERN
NOC = PATTERN
NOV = PATTERN
NVE = PATTERN
O = 2
OP = ARRAY('2,2') #10
OPER = 'EXP((-2IH)X)|(DX-(G1+IH)UDU-(G2+IH)VDV)'
OTN = PATTERN
P = 3
R = 1
REM = PATTERN
RND = PATTERN
S = 1
SCD = PATTERN
SVR = PATTERN
TOT = TABLE(31) #8
TMB = 'G2+IH'
USE = PATTERN
VAR = 'V'
```

DUMP OF KEYWORDS

```
&ABEND = 0
&ANCHOR = 1
&CODE = 0
&DUMP = 2
&ERRLIMIT = 0
&ERRTYPE = 0
&FNCLEVEL = 0
&FTRACE = 0
&FULLSCAN = 0
&INPUT = 1
&MAXLNGTH = 5000
&OUTPUT = 1
&PNTYPE = 'RETURN'
&STCOUNT = 818
&STLIMIT = 50000
&STNO = 144
&TRACE = 0
&TRIM = 1
```

ADDEX #1

```
AS = '+'
AN = 1
AC =
AT = '$$'
AD =
```

EXFUN #2

```
EI = 1
EA = ADDEX #1
```

TABLE(111) #3

```
CMP<0> = ARRAY(4) #4
CMP<11> = TABLE(5) #11
CMP<1> = 'X'
CMP<2> = 'U'
CMP<3> = 'V'
CMP<41> = 'G1'
CMP<42> = 'H'
CMP<43> = 'G2'
```

ARRAY(4) #4

```
<1> = 3
<2> = 11
<3> = 30
<4> = 43
```

TABLE(11) #5

TABLE(21) #6

TABLE(21) #7

TABLE(31) #8

TABLE(3) #9

```
IN<'U'> = 1
IN<'V'> = 2
```

ARRAY('2,2') #10

```
OP<1,1> = EXFUN #13
OP<2,1> = EXFUN #28
OP<1,2> = OPDER #15
OP<2,2> = OPDER #31
```

TABLE(5) #11

```
<1> = 'EXP('
<2> = '(G1+IH)X'
<0> = 3
<3> = '(-2IH)X'
```

ADDEX #12

```
AS = '+'
AN = 1
AC =
AT = 'EXP((G1+IH)X)U'
AD =
```

EXFUN #13

```
EI = 1
EA = ADDEX #12
```

ARRAY('4,3') #14

```
<1,1> = EXFUN #17
<2,1> = 1
<3,1> = 1
<4,1> = 'X'
<1,2> = EXFUN #19
<2,2> = 10
<3,2> = 1
<4,2> = 'U'
<1,3> = EXFUN #23
<2,3> = 100
<3,3> = 1
<4,3> = 'V'
```

OPDER #15

```
OI = 3
OA = ARRAY('4,3') #14
```

ADDEX #16

```
AS = '+'
AN = 1
AC =
AT = 'EXP((G1+IH)X)U'
AD =
```

EXFUN #17

```
EI = 1
EA = ADDEX #16
```

ARRAY(2) #18

```
<1> = ADDEX #20
<2> = ADDEX #21
```

EXFUN #19

```
EI = 2
EA = ARRAY(2) #18
```

ADDEX #20

```
AS = '-'
AN = 1
AC = 'G1'
AT = 'EXP((G1+IH)X)UU'
AD =
```

ADDEX #21

```
AS = '+'
AN = 1
AC = 'IH'
AT = 'EXP((G1+IH)X)UU'
AD =
```

ARRAY(2) #22

```
<1> = ADDEX #24
<2> = ADDEX #25
```

EXFUN #23

```
EI = 2
EA = ARRAY(2) #22
```

ADDEX #24

```
AS = '-'
AN = 1
AC = 'G2'
AT = 'EXP((G1+IH)X)UV'
AD =
```

ADDEX #25

```
AS = '+'
AN = 1
AC = 'IH'
AT = 'EXP((G1+IH)X)UV'
AD =
```

TABLE(5) #26

ADDEX #27

```
AS = '+'
AN = 1
AC =
AT = 'EXP((-2IH)X)'
AD =
```

EXFUN #28

```
EI = 1
EA = ADDEX #27
```

TABLE(5) #29

```
<2> = '-(G1+IH)U'
<3> = '-(G2+IH)V'

ARRAY('4,3') #30

<1,1> = EXFUN #36
<2,1> = 1
<3,1> = 1
<4,1> = 'X'
<1,2> = EXFUN #46
<2,2> = 10
<3,2> = 1
<4,2> = 'U'
<1,3> = EXFUN #58
<2,3> = 100
<3,3> = 1
<4,3> = 'V'

ORDER #31

OI = 3
OA = ARRAY('4,3') #30

TABLE(5) #32


ADDEX #33

AS = '+'
AN = 1
AC =
AT =
AD =

EXFUN #34

EI = 1
EA = ADDEX #33

ADDEX #35

AS = '+'
AN = 1
AC =
AT = 'EXP((-2IH)X)'
AD =

EXFUN #36

EI = 1
EA = ADDEX #35

TABLE(5) #37

<1> = ADDEX #42
<2> = ADDEX #43

TABLE(5) #38

<1> = ADDEX #39
<2> = ADDEX #40

ADDEX #39

AS = '+'
AN = 1
AC = 'G1'
AT =
AD =

ADDEX #40

AS = '+'
AN = 1
AC = 'IH'
AT =
AD =

EXFUN #41

EI = 2
EA = TABLE(5) #38

ADDEX #42

AS = '-'
AN = 1
AC = 'G1'
AT = 'U'
AD =

ADDEX #43

AS = '-'
AN = 1
AC = 'IH'
AT = 'U'
AD =

EXFUN #44

EI = 2
EA = TABLE(5) #37

ARRAY(2) #45

<1> = ADDEX #47
<2> = ADDEX #48

EXFUN #46

EI = 2
EA = ARRAY(2) #45

ADDEX #47

AS = '-'
AN = 1
AC = 'G1'
AT = 'EXP((-2IH)X)U'
AD =

ADDEX #48

AS = '-'
AN = 1
AC = 'IH'
AT = 'EXP((-2IH)X)U'
AD =

TABLE(5) #49

<1> = ADDEX #54
<2> = ADDEX #55

TABLE(5) #50

<1> = ADDEX #51
<2> = ADDEX #52

ADDEX #51

AS = '+'
AN = 1
AC = 'G2'
AT =
AD =

ADDEX #52

AS = '+'
AN = 1
AC = 'IH'
AT =
AD =

EXFUN #53

EI = 2
EA = TABLE(5) #50

ADDEX #54

AS = '-'
AN = 1
AC = 'G2'
AT = 'V'
AD =

ADDEX #55

AS = '-'
AN = 1
AC = 'IH'
AT = 'V'
AD =

EXFUN #56

EI = 2
EA = TABLE(5) #49

ARRAY(2) #57

<1> = ADDEX #59
<2> = ADDEX #60

EXFUN #58

EI = 2
EA = ARRAY(2) #57

ADDEX #59

AS = '-'
AN = 1
AC = 'G2'
AT = 'EXP((-2IH)X)V'
AD =

ADDEX #60

AS = '-'
AN = 1
AC = 'IH'
AT = 'EXP((-2IH)X)V'
AD =
```

```
DUMP OF NATURAL VARIABLES

ABORT = PATTERN
ACN = PATTERN
ANV = PATTERN
BAL = PATTERN
BPR = PATTERN
CCE = TABLE(11) #5
CMP = TABLE(11) #3
CNA = PATTERN
CNP = PATTERN
CNS = PATTERN
DTA = TABLE(21) #7
FAO = EXFUN #2
FCE = TABLE(21) #6
FENCE = PATTERN
IN = TABLE(3) #9
INPUT = 'CHEK'
J = 1
K = 29
L = 3
NCN = PATTERN
NOC = PATTERN
NOV = PATTERN
NVE = PATTERN
O = 2
OP = ARRAY('2,2') #11
OPER = '(X-X@(3))/(R)DXDX'
OTN = PATTERN
P = 1
Q = 1
REM = PATTERN
RNO = PATTERN
S = 1
SCO = PATTERN
SVR = PATTERN
TDT = TABLE(31) #8
TMR = 'X-X@(3)'
USE = PATTERN
VAR = 'X'

DUMP OF KEYWORDS

&ABEND = 0
&ANCHOR = 1
&CODE = 0
&DUMP = 2
&ERRLIMIT = 0
&ERRTYPE = 0
&FNCLEVEL = 0
&FTRACE = 0
&FULLSCAN = 0
&INPUT = 1
&MAXLNGTH = 5000
&OUTPUT = 1
&RTNTYPE = 'RETURN'
&STCOUNT = 1131
&STLIMIT = 50000
&STNO = 144
&TRACE = 0
&TRIM = 1

ADDEX #1

AS = '+'
AN = 1
AC =
AT = '$$'
AD =

EXFUN #2

EI = 1
EA = ADDEX #1

TABLE(11) #3

CMP<0> = ARRAY(4) #4
CMP<11> = TABLE(5) #10
CMP<1> = 'X'
CMP<2> = 'U'
CMP<3> = 'V'
CMP<41> = 'G1'
CMP<42> = 'H'
CMP<43> = 'G2'
CMP<4> = 'R'
CMP<31> = '(1-X@(2))'

ARRAY(4) #4

<1> = 4
<2> = 11
<3> = 31
<4> = 43

TABLE(11) #5

TABLE(21) #6

TABLE(21) #7

TABLE(31) #8

TABLE(3) #9

IN<'U'> = 1
IN<'V'> = 2
IN<'R'> = 3

TABLE(5) #10

<1> = 'EXP('
<2> = '(G1+IH)X'
<0> = 3
<3> = '(-2IH)X'

ARRAY('2,2') #11

OP<1,2> = OPDER #14
OP<2,2> = OPDER #22

TABLE(5) #12

<1> = '-I(1-X@(2))@(-.5)'
<2> = '-.5I(1-X@(2))@(-.5)'

ARRAY('4,2'1 #13

<1,1> = EXFUN #16
<2,1> = 1100
<3,1> = 2
<4,1> = 'RV'
<1,2> = EXFUN #19
<2,2> = 100
<3,2> = 1
<4,2> = 'V'

OPDER #14

OI = 2
OA = ARRAY('4,2') #13

ADDEX #15

AS = '-'
AN = 1
AC = 'I'
AT = '(1-X@(2))@(-.5)'
AD =

EXFUN #16

EI = 1
EA = ADDEX #15

TABLE(5) #17

ADDEX #18

AS = '-'
AN = '.5'
AC = 'I'
AT = '(1-X@(2))@(-.5)'
AD =

EXFUN #19

EI = 1
EA = ADDEX #18

TABLE(5) #20

<1> = '(X-X@(3))/(R)'

ARRAY('4,1'1 #21

<1,1> = EXFUN #33
<2,1> = 2
<3,1> = 2
<4,1> = 'XX'

OPDER #22

OI = 1
OA = ARRAY('4,1') #21

TABLE(5) #23

<1> = ADDEX #31
<2> = ADDEX #32

TABLE(5) #24

ADDEX #25

AS = '+'
AN = 1
AC =
AT = 'R'
AD =

EXFUN #26

EI = 1
EA = ADDEX #25

TABLE(5) #27

<1> = ADDEX #28
<2> = ADDEX #29

ADDEX #28

AS = '+'
AN = 1
AC =
AT = 'X'
AD =

ADDEX #29

AS = '-'
AN = 1
AC =
AT = 'X@(3)'
AD =

EXFUN #30

EI = 2
EA = TABLE(5) #27

ADDEX #31

AS = '+'
AN = 1
AC =
AT = 'X'
AD = EXFUN #26

ADDEX #32

AS = '-'
AN = 1
AC =
AT = 'X@(3)'
AD = EXFUN #26

EXFUN #33

EI = 2
EA = TABLE(5) #23
```

At the start of the program certain keywords are set to allow for necessary non-default processing modes. &STLIMIT is set at 200,000, since, with an average time per statement of about 0.4 msec, the default value of 50,000 allows for only 20 seconds of processing. The time limit for the standard run is 55 seconds*. &ANCHOR is set to 1 and kept there except in places where it is switched to zero for certain short sequences. &DUMP is set to 2 at the start and is changed back to zero only if the program terminates normally; thus errors sufficient to force termination also force a detailed listing of the information stored at that time.

The termination sequence (line 1268) may be modified to provide any special type of normal termination, such as one which includes a dump.

Below we examine the specific operations of the various p-functions in the program. As aids in understanding how the p-functions fit into the program as a whole, a tally, p. 120, of the p-variables used in each p-function, a program flowchart, Fig. 5.3.3, p. 122, and an accompanying diagram illustrating the nesting of the p-functions, Fig. 5.3.4, p. 123, appears at the end of this section; reference to them during the reading of the following part of this

* This is set in the job control language statements (lines 1 to 6); see Section 6.3, p. 129, for details.

section may be helpful in understanding the descriptions.

The p-functions are defined in lines 20 to 49 of the program listing (Appendix B). They are described below in that order, which is the same as the order in which the p-function routines themselves appear in the program listing. The name of the p-function is written in the margin at the start of the discussion of it.

SQEZ1 extracts variables from the input strings; it is called in lines 87, 152 and 190 as a part of the input sequence. Statements labelled VRSQE and CNSQE (lines 207 and 215) contain the most interesting steps in SQEZ1. In them the input string (assigned to p-variable OPER) remains unchanged as the variables and constants are picked out of it and stored. The pattern NVE (see line 52) causes the cursor to skip everything except variables and the "useful functions" (exp, sin, cos, lgn, and sqt) in the search for variables. If the pattern NVE fails to match, pattern NOV (line 53) is tried. NOV causes the cursor to also skip the names of the "useful functions" and skip the contents of their arguments which are not variables. NOV can also match the null string. The initial position of the cursor is set by the value of the p-variable Q - to the right of a just-examined variable. Patterns NCN and NOC perform analogous operations in the search for constants.

SQEZ1
line 204

Since SQEZ1 acts only on each line of input, breaking a subscripted constant such as "A23" at the end of a line would result in the wrong constant being stored in table CMP ("A" or "A2" here). Blanks and asterisks are removed in SQEZ1 also (line 205, labelled RMBLK). See also the discussion of SQEZ1 in connection with cannoni-

cal ordering in Section 5.2, page 89.

In the first line of CONOP (line 223), parentheses are removed from the operator. In lines 226 and 227 the number of additive operator components P is determined. Each is assigned to an element of table TBL. P-variable CONOP is then given the form OPDER(P,ARRAY(4,P)), in which is stored relevant information about the operator. Elements OA(CONOP)<3,K> and OA(CONOP)<4,K>, where K is the index of one of the additive operator components, are assigned within p-function DINFO, called in line 232. In the same line OA(CONOP)<2,K> is also assigned. These three elements contain information about the Kth set of derivative coefficients. The function associated with the Kth additive operator component is converted to datatype EXFUN, multiplied by the operator's common left multiplier, and assigned to element OA(CONOP)<1,K> in line 234. These assignments are also discussed in Section 5.1, p. 84.

CONOP line 223

CONOP is called in lines 102 and 109 as part of the input sequence. Once operators are converted to datatype OPDER, there is no further need for CONOP.

As is mentioned above, two of the four array elements of an additive operator component are assigned within DINFO. The structure of the lexical ordering number, one of these two elements, is shown in line 245 (it is also described in Section 5.1, pp. 84 and 85).

DINFO line 243

P-function CONEX is used whenever it is necessary to convert a function written as a string to datatype EXFUN. This occurs when

CONEX line 250

the operator's common left multiplier is converted in the input sequence, when the same distributively multiplies the rest of the operator, the functions associated with the additive operator components of which must be in the same form, when a function is to have an operator applied to it via "APLY", and within the derivative p-function DERAD. In the latter, arguments to the "useful functions" and parenthesized factors sometimes require conversion. CONEX is also called within CONEX itself to carry out the expansions of functions, when necessary, and to convert a non-null AD field.

CONEX can be divided into four similar sequences, starting with lines 254, 278, 304 and 329, respectively. For each, an additive component is extracted from a function expressed as a string, and the component is broken down into elements which are identified with fields AS, AN, AC, AT and AD of datatype ADDEX (see also p. 87). Following this, the fields are altered to put them in standard form.

If there is no numerical multiplier, the AN field is set to 1. With this in mind, we note that, say, "++X" converts the same as "+1+X" or "1+X" and is worth either of the latter in the computations. If the AS field is null, it is changed to "+". The AD field, if non-null, is converted to datatype EXFUN, and the AC and AT fields are changed when functions are expanded due to the presence of parenthesized factors in them. For the expansion of functions with parenthesized factors in the AT fields, the pattern SVR, used in lines 261, 287, 311 and 336, skips over parenthesized expressions raised to a power and other functions which may be found in the AT field, except

only for parenthesized factors not raised to a power, which are identified by the remainders of the statements in those lines. The expansion of functions with parenthesized factors in the AC fields is similar to but less complex than the above due to the relative simplicity of constant expressions.

P-function SQEZ2 is called in p-function CONEX. It serves to store, in the table CMP, the "useful functions" and parenthesized factors raised to a power. Since SQEZ2 analyzes the AT fields of functions in CONEX before the expansion due to parenthesized factors in them, the pattern SFU used in line 385 is designed to ignore parenthesized factors which are not raised to a power; the latter are lost in the subsequent expansion and may not, therefore, appear in the results. Other details of the operation of SQEZ2 are given in Section 5.2, p. 90.

SQEZ2 line 355

As with SQEZ1, the value of the p-variable FCT does not change as a result of the analys s performed; the cursor position is also set by the value of p-variable Q here. Since there may be relevant information in the arguments, they are analyzed also, e. g., the argument of "SIN(Y-(1-X@(2)@(.5))" contains the factor "(1-X@(2))@(.5)".

MULPO performs multiplication for functions written in data-type EXFUN. There are four similar segments in MULPO, beginning, respectively, with lines 399, 405, 411 and 418. The multiplication of additive components of functions is basically a one-step process in which concatenation and shortening of the pair of AS fields,

MULPO line 394

multiplication of the pair of AN fields, concatenation of the AC and AT fields, and multiplication of the AD fields (via MULPO) are all performed in one statement. The differences between the four segments result from the manner in which the additive components are referenced: when the EI field is equal to 1, the EA field is of datatype ADDEX; otherwise the EA field is an array. The four possibilities of pairs of functions, (EI_1,EI_2) = (1,1), (1, not 1), (not 1, 1) and (not 1, not 1), correspond to the four segments. When one of the functions is null, the product is not zero but the other function. For convenience, since much of the product routine is done by concatenation rather than by numerical-type multiplication, the null string serves to represent the number 1 (and yields the same result as multiplication by EXFUN(1,ADDEX("+",1,,,)) which is the number 1, converted). A non-zero number must always be present in field AN of datatype ADDEX.

The p-variable SGN, seen in MULPO, finds use in p-function OPCON.

SGNCO shortens a string of signs, encountered in function multiplication, to a single equivalent sign; thus "+++" is changed to "+", and "-+--", to "-". SGNCO line 427

In the first line of p-function OPCOM, an array is created which has room for every possible product of additive operator components in the commutation of the operators referenced by the formal p-function arguments OPA and OPB (written in datatype OPDER). An element referenced by zero is also provided to store the size of OPCOM line 431

the array for use later.

The p-variable N is set to control processing in the last stage of p-function CYHAC, which follows this. The variables referenced in the call of CYHAC are determined by the patterns in lines 436 and 450, which pick them out by threes from the string set up by p-function DINFO as a part of the conversion done by CONOP. In the first half of OPCOM (to line 446), the OPA·OPB part of [OPA,OPB] is determined; OPB·OPA is found in the second half. Setting SGN equal to "-" in the second half changes the signs on all the terms in the latter operator product.

Considering the third-order commutator of additive operator components OPA and OPB: CYHAC line 463

$$
\begin{aligned}
[OPA,OPB] &= [f(u,v,w)\partial_x\partial_y\partial_z, g(x,y,z)\partial_u\partial_v\partial_w] \\
&= f\{g\partial_x\partial_y\partial_z + g_x\partial_y\partial_z + g_y\partial_x\partial_z + g_z\partial_x\partial_y + g_{xy}\partial_z \\
&\quad + g_{xz}\partial_y + g_{yz}\partial_x + g_{xyz}\}\partial_u\partial_v\partial_w \\
&\quad - g\{f\partial_u\partial_v\partial_w + f_u\partial_v\partial_w + f_v\partial_u\partial_w + f_w\partial_u\partial_v + f_{uv}\partial_w \\
&\quad + f_{uw}\partial_v + f_{vw}\partial_u + f_{uvw}\}\partial_x\partial_y\partial_z *,
\end{aligned}
$$

the action of p-function CYHAC can neatly be represented by a diagram (Fig. 5.3.1, next page).

The p-variable S, set equal to 1 at the start of the program, is increased when the p-function is not operating with the last of

* g and f represent explicit functions. The subscripts indicate differentiation.

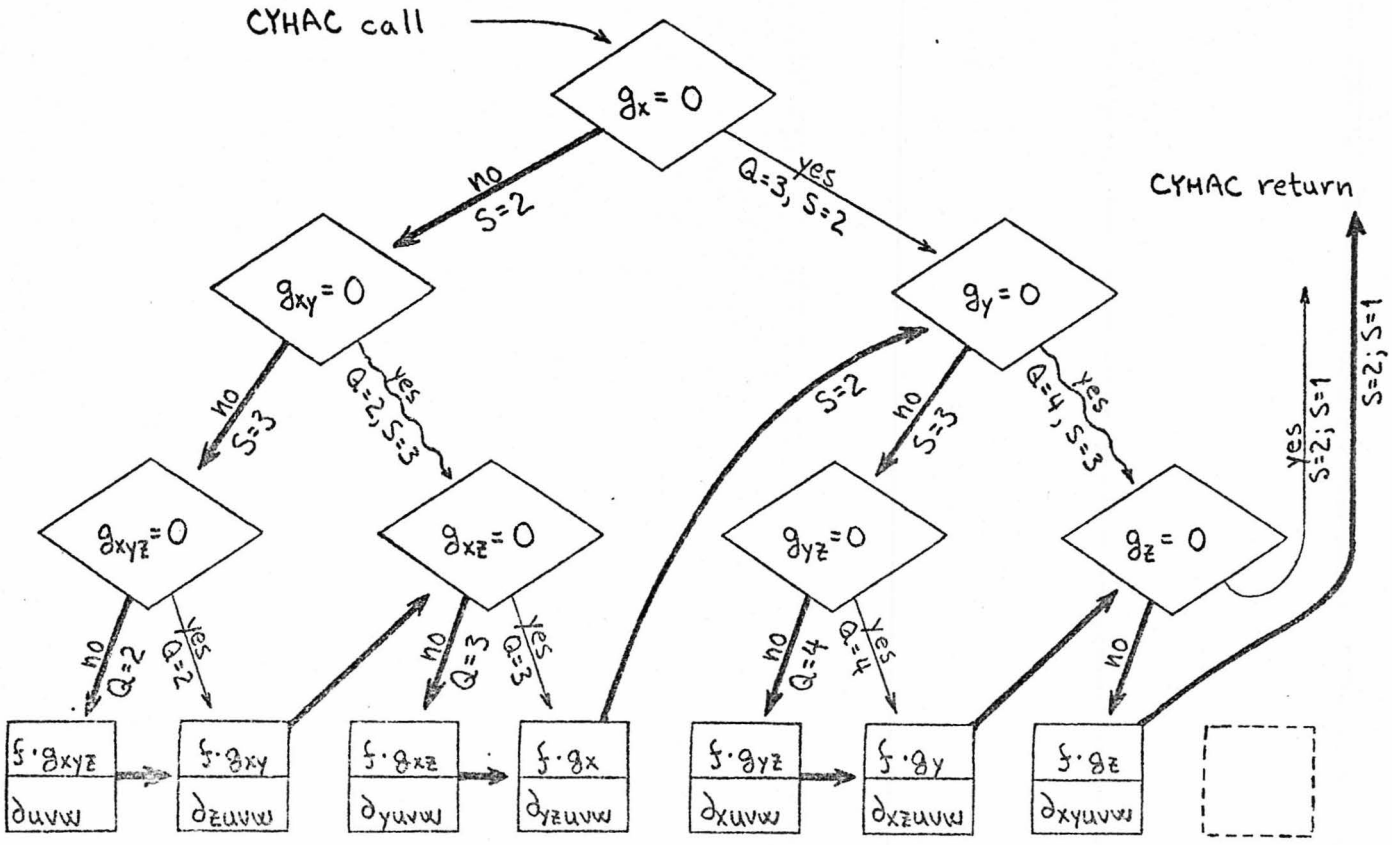

Figure 5.3.1 Diagram of the product $f\partial_x\partial_y\partial_z \cdot g\partial_u\partial_v\partial_w$ (half of the commutator), as performed by p-function CYHAC. The diamonds symbolize the analyses performed on the second operator to determine what terms of the product are to be saved (put in the boxes). For $g = g(x,y,z)$ the heavy lines show the route of processing; if instead, say, $g = g(x,z)$, the wavy shortcuts are taken. The term $fg\partial_x\partial_y\partial_z\partial_u\partial_v\partial_w$ is not generated; it would cancel with a corresponding term of the product $-g\partial_u\partial_v\partial_w \cdot f\partial_x\partial_y\partial_z$, which is not generated either.

the p-variables VAR, VR2, VR3, ..., VR9. In this manner the p-function keeps track of the level on which it is operating. When the p-function reaches the level in which it is operating with the last

of the p-variables, it begins the storage of the data which is later collected, as required, in the lower levels of the p-function operation. The value of S is decreased as the p-function returns through the levels. The values of S are recorded on Fig. 5.3.1.

In a product of additive operator components, the total number of additive operator components in the result with distinct sets of derivative coefficients depends, in general, on the order of the first component of the product, e. g., in the product given above, operator component OPA has order 3, and the result of OPA·OPB has 8 components with distinct sets of derivative coefficients. If the order of OPA is m, the number of components in the result is, in general, 2^m. In p-function OPCOM the value of p-variable N is set (lines 435 and 449) to reflect this maximum (N is actually 2^{m-1}). In CYHAC the value of Q is compared with N to determine whether the point has been reached to discard the term common to the products OPA·OPB and OPB·OPA. Figure 5.3.1 shows the values of Q and indicates the excluded term with a dotted box. Note that the changes in Q along the branches descending to the right depend on the value of S and on the order of OPA (see line 466).

The operator product and anticommutation p-function OPMUL could have been included into the commutation p-function OPCOM by keying certain expressions to behave differently, but the commutation p-function is likely to see the most use and thus should be as "clean" as possible. The p-function OPMUL, being separate, is also "clean" and is somewhat less complex than OPCOM, considering the

OPMUL
line 511

comparison of CYMUL (discussed below) and CYHAC also.

In OPMUL, if the multiplying operator is of order zero, the result is calculated right away, without calling CYMUL. Such terms are omitted when they occur in OPCOM since they always cancel with some other term also omitted.

Neither of the p-variables N and Q, used in OPCOM, are needed in OPMUL.

CYMUL behaves almost like CYHAC, except for the inclusion of the term omitted in CYHAC. Thus Fig. 5.3.1, p. 104, also represents CYMUL, except that the contents of the dotted box (fg and $\partial_x\partial_y\partial_z\partial_u\partial_v\partial_w$) is added to the data before the return of the function and that the p-variable Q is not used. CYMUL line 544

In p-function OPAPL the program performs what may be considered the action of an operator of some order on an operator of zeroth order, where only one component of the product is retained, e. g., for $f\partial_x\partial_y\partial_z \cdot g$, the term retained is $f \cdot g_{xyz}$. There are no derivative coefficients associated with the result of the application of an operator to a function. This simplification allows OPAPL to be written quite briefly. OPAPL line 583

P-function CYAPL, in which the derivative is cyclically performed, is also a short p-function. For the result to be non-zero, the function must be a function of all the variables appearing in the derivative coefficients of the operator. CYAPL line 595

An example in which OPMUL and OPAPL are called is Example 3, Chapter IV, pp. 49 to 60.

In p-function DERAD we see the first use of tables to save execution time by storing the results of complex operations which must be performed more than once. In the first line of DERAD, the value of the p-variable ARG, the argument for table references, is computed. Since functions embedded in datatypes EXFUN and ADDEX, though they may be the same functions, are considered distinct unless previously made equivalent (using "="), the references, say, TDT<FA> and TDT<FB>, where FA = EXFUN(1,ADDEX("+",2,,"X@(2)",)) and FB = EXFUN(1,ADDEX("+",2,,"X@(2)",)), will each give a value EXFUN(1, ADDEX("+",4,,"X",)) (the derivative), but the two results are not considered equivalent. They were created at different times, and no connection exists between them. Allowing the value of ARG, then, to refer to objects such as FA and FB is of no help in building a table of derivatives, where such arguments must be recognized as equivalent. ARG is thus set equal to the string value of the function (with the variable of differentiation appended), and the table serves its intended purpose.

DERAD
line 604

Table TDT is a temporary table of derivatives. The value of TDT<ARG> is the derivative, in datatype EXFUN, of the function whose string value gas been assigned to p-variable ARG. When the function represented by ARG has not recently or has never been differentiated, the value of TDT<ARG> is null, and, unless in the former case the derivative is stored in the permanent derivative table DTA under DTA<ARG>, the p-function DERAD will determine the derivative of the

function and assign it to TDT<ARG>. To prevent the possible necessity of storing excessively long strings for use as referencing arguments in TDT and DTA, functions with string equivalents of length greater than or equal to 60 characters may not have entries in the tables. See lines 604 to 607.

In order to qualify to be an entry in table DTA, the permanent derivative table, a function must be differentiated twice out of 30 (at least) consecutive differentiations performed. After 30 entries have been made in table TDT, it is regenerated via p-function REGEN (described below - p. 111). The last ten entries become the first ten entries in a newly-created table TDT; after 30 more elements are added to TDT, the table is regenerated again, and so forth.

When the derivative is not found in the tables, the rest of DERAD works on the function. First, an additive component is separated from the function if it has more than one. The component is analyzed for the presence of divisors; if one is present, its derivative is taken. For a function with neither additive components nor divisors, all that remains is the differentiation of the AT field.

Since the divisors do not themselves have divisors but may have additive components, the analysis of AT fields is never more than three levels away from any function presented for differentiation. Consider the function $y + 1/(1 - x)$. To differentiate this with respect to, say, x, the program first separates the $+1/(1 - x)$ and

differentiates it. In so doing the p-function recognizes that the function +1/(1 - x) has an EI field equal to 1; therefore DERAD proceeds to the next test, the check for divisors. Since the function meets the requirements (non-null AD field), the divisor is differentiated; DERAD is thus presented with the function 1 - x, a function with additive components. The function -x is separated and differentiated next; it has neither additive components nor divisors. Differentiation of -x is thus effectively performed by knowing its AT field ("X"). When DERAD returns the value of the derivative of -x, the p-function proceeds to differentiate the next additive component of 1 - x after -x, which is 1. Whenever a derivative is zero, p-function DERAD fails; here the operation of differentiating the function 1 results in a failure of DERAD. Such failure is used to directly control the sequence of operations through the use of failure goto branches. When the value of the derivative of 1 - x is returned, the p-function differentiates the numerator of the function +1/(1 - x). Failure of DERAD here results in the return of the derivative of 1 - x times 1/(1 - x - x +xx) as the derivative of +1/(1 - x). When this is known, DERAD proceeds to the next step in the evaluation, the differentiation of the additive component which comes after +1/(1 - x) in y + 1/(1 - x), y; this derivative is zero. DERAD fails, and the value returned as the derivative of y + 1/(1 - x) is the same as the value returned for the derivative of +1/(1 - x).

In the analyses of the AT fields, the function component is

tested early for the presence of variables (line 628). If variables are not present, either the derivative is zero or is undefined. If it is zero, the p-function fails. The derivative is undefined when some function, as a result of a previous operation, was assigned the value "$$". This happens when a function contains a variable but cannot be differentiated, e. g., "X@(2X)". The derivative of "$$" is "$$" and is also undefined. When an error occurs in DERAD, a message indicates the problem, and the printout is keyed with "$$".

The function is analyzed to see: (1) if the AT field contains only the variable (line 631), in which case the derivative is quite simple; (2) whether the AT field contains factors which are powers of the variable (line 635); (3) if the AT field begins with factors (variables other than the variable of differentiation, parenthesized factors, or one of the "useful functions") raised to a power (line 703); (4) if the AT field begins with one of the "useful functions" (exp, sin, cos, lgn and sqt) not raised to a power (line 748); (5) whether the AT field begins with "@" or "/" due to errors in format (line 777); (6) whether the AT field can be broken down into two balanced factors (line 779); and, finally, (7) if the AT field is some parenthesized expression, the contents of which may be differentiated (line 789).

The p-function DERAD appears as the main component of the program available under the filename p799.math.deriv.add on sys08 in the Stanford library of math-oriented programs.

P-function FUNST is a primitive conversion p-function which serves the purpose of quickly putting functions into string format for use as referencing arguments in tables TDT and DTA. It is called at the start of p-function DERAD. FUNST line 805

ARCHA creates and fills an array required when two functions written in datatype EXFUN are added together. ARCHA is called at the end of DERAD. ARCHA line 818

VARID checks the AT fields of each component of a divisor for the presence of variables (via VARIN - see below). If a component contains a divisor, in violation of format restrictions, its presence is noted by VARID in the printout. VARID line 834

P-function VARIN checks a string of characters for the presence of the variable of differentiation to determine whether further processing is necessary. If the variable is "X" or "T", respective occurrences of "EXP(" or "SQT(" are replaced by the null string before the check is finally verified. VARIN line 846

In REGEN, which regenerates the table TDT, the table is first converted to an array. The last 10 elements of this array are assigned to the newly created table REGEN. REGEN line 854

P-function SIMCO rearranges the result of OPCOM, OPMUL or OPAPL so that it is ordered and simplified on output. The first part of SIMCO performs the ordering. Following this is the display of the "ordered result". Then the simplification is performed, followed by the display of the "simplified result". SIMCO line 861

For the ordering, p-variable SIMCO is set equal to a 10-element vector array with indices from 0 to 9. Each element of this array is eventually associated with all the additive components of order equal to the index of the element. Thus SIMCØ⟨0⟩ would be the additive operator components of order zero, and SIMCO⟨M⟩, of order M. The (at most) ten array elements, if non-null, would each be written in datatype OPDER, the OA field being ARRAY(2,K), where K is the number of distinct sets of derivative coefficients of order M.

The first step in setting up such an agreeable arrangement is the counting of the number of additive operator components I_M associated with each order M (remember that, at the start, these can be scattered all over - see lines 865 to 872). Using this information, the program sets up arrays SIMCØ⟨M⟩ = ARRAY(2,0:I_M), $I_M > 0$ (as an exception, SIMCØ⟨0⟩ = ARRAY(0:I_0), $I_0 > 0$), lines 875 to 877, in which the functions (written in datatype EXFUN) and the sets of derivative coefficients (written as lexical ordering numbers) associated with each additive operator component of order M are stored in respective elements SIMCO⟨M⟩⟨1,N⟩ and SIMCO⟨M⟩⟨2,N⟩, $1 \leq N \leq I_M$, $M > 0$ (for zeroth-order components, the functions are stored in elements SIMCØ⟨0⟩⟨N⟩, $1 \leq N \leq I_0$). SIMCØ⟨M⟩⟨1,0⟩ (or, for M = 0, SIMCØ⟨0⟩⟨0⟩) contains the value of I_M (I_0) at the end of this procedure. See lines 879 to 887.

As the next step in the ordering procedure, a table OPER = TABLE(10) is created (note that M is not incremented until the end of the processing in the next paragraph). Each element of table OPER is

a three element vector array (OPER<Q> = ARRAY(3)). A lexical ordering number is stored in OPER<Q><1>, and the number and identities of all the functions associated with it are stored in respective elements OPER<Q><2> and OPER<Q><3>. OPER<Q><3> is, in turn, a 10-element table; its entries are the indices J for all SIMCO<M><2,J> (see the previous paragraph) which equal OPER<Q><1>. Further, for Q > R, OPER<Q><1> < OPER<R><1>. Thus, in table OPER, each distinct set of derivative coefficients of order M is ordered by the lexical ordering numbers in reverse numeric sequence, and every function associated with each set is readily identifiable. When this procedure (lines 898 to 918 and 940 to 954) finishes, the number of distinct sets of derivative coefficients (the maximum value of Q) is stored in array element SIMCØ<M><2,0>.

The final step in the ordering (lines 921 to 936 and 956 to 973) is the rewriting of the array elements SIMCO<M> as SIMCO<M> = OPDER(S,ARRAY(2,S)), where S = SIMCØ<M><2,0>, the number of distinct sets of derivative coefficients. Adding the EI fields of the functions, we arrive at a number N_Q which can be used to set up a structure in datatype EXFUN, namely EXFUN(N_Q,ARRAY(N_Q)), which can be assigned to OA(SIMCO<M>)<1,Q> and which can be filled up by components of the functions identified by the table OPER<Q><3>. Each element OA(SIMCO<M>)<1,Q> is then the sum of the functions identified in table OPER<Q><3>. OA(SIMCO<M>)<2,Q> is, of course, the lexical ordering number previously identified by OPER<Q><1>. M is incre-

mented here (M → M - 1), and the program begins the processing of the preceding paragraph again for the new value of M, except when M = 0, in which case the following paragraph applies.

For SIMCØ<0> there is no set of derivative coefficients, so the procedure involves only the combination of the functions into one EXFUN structure (lines 977 to 990). SIMCØ<0> has the final structure OPDER(1,ARRAY(2,1)), where EA(SIMCØ<0>)<2,1> is null, to allow for output processing by p-function OPOUT (see below).

After the "ordered result" is displayed (lines 1001 to 1006), the result is simplified. Each function associated with an additive operator component is cannonically ordered (lines 1010 to 1026), then simplified (line 1028).

The AC (constant) field of an additive component of a function is cannonically ordered first, followed by the same on the AT field. The AC field is also checked for the presence of powers of the constant "I" (lines 1013 to 1019), which is interpreted to be $\sqrt{-1}$; the AS (sign) field is adjusted accordingly. All the additive components of a function are sequentially changed in this manner. For cannonical ordering of the AC and AT fields, p-functions CONCA and FUNCA (see below) are respectively called (lines 1022 and 1025).

For simplification, p-function ADSUB (see below) is called (line 1028). When a function associated with an additive operator component is found to be zero, the p-function ADSUB fails. This information causes the program to shift the other components of the

same order to fill in the gap or to take action on the finding that the entire set of components of the same order is zero (lines 1034 to 1039). If, as a result of the simplification, the entire operator is zero, the p-function SIMCO fails and "NULL RESULT" is displayed (this occurs in the input sequence, from which SIMCO is called - see lines 139 to 142, 155 to 158, 184 to 187, and 193 to 196). For non-null operators the "simplified result" is displayed as a result of a second call of p-function OPOUT (lines 1048 to 1052).

OPOUT converts an operator, written as an array of elements each of datatype OPDER, to a string format which resembles the input format in all details except in the use of the symbols "<>" as angular brackets and the double and triple spacing between, respectively, components of the same order and those of a different order. these latter are concessions to readability. OPOUT line 1055

In OPOUT strings which are components of functions are concatenated until the length of the resulting string is such that adding another component would cause the length to become greater than 90 characters (93 characters for the second and subsequent lines). If the addition of the extra component causes the length to become less than 105 characters (108 characters for the second and subsequent lines), the string with the added component is displayed; otherwise the string is displayed without the added component.

P-function DVOUT converts divisors from datatype EXFUN to strings for use in p-function OPOUT. If the divisor contains any DVOUT line 1112

AD (divisor) fields in violation of format restrictions, a note of this is made on the printout.

ONTDC converts a lexical ordering number to a string which is either a set of derivative coefficients or a string of the variables in the set, depending on whether the formal argument DR is set to "D" or to null. ONTDC line 1133

The conversion of constant fields AC to cannonical form is performed in p-function CONCA. Previous results are stored in tables PCT and CCF in a manner analogous to the use of tables TDT and DTA by p-function DERAD (p. 107). CONCA line 1137

In lines 1153 to 1158 is a sequence which may be of some use in p-functions DERAD and FACEX, where powers of a function or variable are similarly combined (see lines 638 to 670 and 1212 to 1223). When testing CONCA before adding the sequence to the program, the field "GHG@(.5)J@(1.33)G@(-.5)JKJ@(-.15)" converted to "GHJ@(2.179999)K"; with the sequence, it converts to "GHJ@(2.18)K". Repeated occurrences of the former could drastically lengthen the output. The sequence serves as a model for extensions to p-functions DERAD and FACEX, added when a user finds that he must use fractional powers extensively.

The sequence in lines 1153 to 1158 operates by rounding off a real number ending in at least two nines (to the right of the decimal point), for real numbers initially written with no more than three digits past the decimal point.

When an expression arrears in the constant field which, as a result of some format error, does not fit into the required form and cannot be converted to cannonical form, a message indicates the failure. The expression retains its form. Further occurrences of the same are so processed without notice.

P-function FUNCA converts the AT fields to cannonical form. Its behavior is similar to that of CONCA. Tables PFT and FCF store previous results, and failure to convert results in the same procedure. FUNCA line 1175

Three types of functions are processed by FUNCA, and they are ordered as follows: variables, the "useful functions", and parenthesized factors. As with CONCA, the ordering called cannonical depends on the ordering of the discrete components (constants, variables, "useful functions", or parenthesized factors) in the input. The ordering is, in fact, the same as that in the input, except for the ordering of the variables, which is reversed; thus "UVSIN(2X)EXP(2V)" as the first occurrences of the discrete components "U", "V", "SIN(2X)", "X" (from the argument of the last), and "EXP(2V)" forces the conversion of, for example, "U@(2)X@(2)UEXP(2V)USIN(2X)@(2)V" to be "X@(2)VU@(4)SIN(2X)@(2)EXP(2V)". Example 4, Chapter IV, pp. 60 to 68, is an interesting study of this mechanism, though the ordering of the variables there is not the same as in the current version of the program.

To keep the size of FUNCA small, the p-function FACEX performs the actual evaluation of the presence of the factors. FACEX line 1209

When a function associated with an additive operator com- ADSUB line 1226

ponent is not expressed as a sum of additive components, its structure should be EXFUN(1,EA), where the EA field has a structure ADDEX(AS,AN,AC,AT,AD). During the ordering process in SIMCO, however, such a function is given instead the structure EXFUN(1,EA), where the EA field is an array ARRAY(1) whose one element has the structure ADDEX(AS,AN,AC,AT,AD). The former structure is required if more is to be done (as under the action of "SAVE") with the operator. Since each function associated with an additive operator component is referenced in SIMCO and since some value must be given to p-variable ADSUB, all one-component functions are changed to the usual form in p-function ADSUB, assigned to p-variable ADSUB, and immediately returned. If other evaluation reduces a larger function to one term, it is also changed, assigned to ADSUB, and returned.

In ADSUB each additive component is compared with each other additive component in an effort to combine common terms. The method of action is shown in Fig. 5.3.2. The maximum number of comparisons

Figure 5.3.2 Illustration of the action of the p-function ADSUB. The boxes represent array elements in the structure EXFUN(N,ARRAY(N)). When two terms I and J, $1 \leq I < J < N$, may be added (when their AC, AT and AD fields are identical), the Jth term is discarded and replaced by the Nth term (the AS and AN fields of the Jth term are saved, to be added to those of the Ith term). The function structure then becomes EXFUN(N - 1,ARRAY(N)), but this is effectively EXFUN(N - 1,ARRAY(N - 1)), since incrementing is controlled by the EI field. When J = N and the terms I and J may be added, the structure changes as indicated, but the replacement is, of course, omitted.

for a function with N components, when no common terms are present, is then $\frac{1}{2}(N^2 - N)$, the sum of 1, 2, ..., (N - 1). Generally, cancellations and combinations reduce this number considerably, however.

As was indicated in Section 5.1, p. 86, RESET puts the result in a form, when required, which corresponds to that of initially converted operators. In this form they may appear as arguments to p-functions OPCOM, OPMUL or OPAPL.

To aid in the understanding of the manner in which the p-variables are used in the p-functions, the tally following on the next two pages is included. This is followed by a chart, Fig. 5.3.3 of the operation of the entire program and a diagram, Fig. 5.3.4, of certain structures referenced in Fig. 5.3.3.

P-VARIABLES TALLY

P-FUNCTION	ARGUMENTS	LOCAL P-VARIABLES	OTHER P-VARIABLES USED IN P-FUNCTION	
			SET INSIDE P-FUNCTION	SET OUTSIDE P-FUNCTION
(INPUT SEQUENCE)	-	-	ANV, RNO, NVE, NOV, CNP, CNA, CNS, NCN, NOC, ACN, SCO, USF, SFU, SVR, OTN, FAD, CMP, CCF, FCF, CC, F, PCT, PFT, J, K, TDT, DTA, S, IN, OUTPUT, OP, O, INPUT, TMA, OPER, FCT	CC, F
SQEZ1	OPER	-	Q, SQEZ1, TMA, P	NVE, NOV, ANV, NCN, NOC, ACN
CONOP	OPER	TBL	P, CONOP, OUTPUT	O
DINFO	DR	-	VAR, DINFO	ANV
CONEX	FCT	AS, AN, AC, AT, AD, P, SI	CONEX, Q, TMA, TMB	RNO, CNS, SCO, SVR
SQEZ2	FCT	P, Q, TMA, ARG	L	USF, SFU
MULPO	TMA, TMB, SGN	A, B	MULPO	-
SGNCO	SGNCO	-	-	-
OPCOM	OPA, OPB	A, B	OPCOM, C, SGN, Q, N, VAR, VR2, VR3, ..., VR9	-
CYHAC	OPA, OPB, A, B, VAR, VR2, VR3, ..., VR9	DR, TMA, TMB	S, Q, CYHAC, M, L	S, Q, N
OPMUL	OPA, OPB, SI	A, B	OPMUL, C, VAR, VR2, VR3, ..., VR9	-
CYMUL	OPA, OPB, A, B, VAR, VR2, VR3, ..., VR9	DR, TMA, TMB	S, CYMUL, L, M	S

P-VARIABLES TALLY
(continued)

P-FUNCTION	ARGUMENTS	LOCAL P-VARIABLES	OTHER P-VARIABLES USED IN P-FUNCTION	
			SET INSIDE P-FUNCTION	SET OUTSIDE P-FUNCTION
OPAPL	OPA, FCT	-	OPAPL, A, VAR, VR2, VR3, ..., VR9	-
CYAPL	OPA, FCT VAR, VR2, VR3, ..., VR9	DR	S, CYAPL	S
DERAD	FCT	DR, ARG	VAR, DERAD, OUTPUT, J, TDT	FAD, ANV, USF, J
FUNST	FCT	P	FUNST	-
ARCHA	TMA, TMB, SI	P, Q	ARCHA, OUTPUT	-
VARID	FCT	-	P, OUTPUT	-
VARIN	FCT	-	-	VAR
REGEN	TBL	-	REGEN, P, K, J	K
SIMCO	OPER	-	SIMCO, P, N, C, Q, M, SI, L, OUTPUT, DR, I, TMA, TMB, CCF, CC, FCF, F	CC, F
OPOUT	OPER	-	M, Q, OUTPUT, TBL, TMA, TMB, P, OPOUT	C
DVOUT	FCT	P	OUTPUT, TMA, TMB, SI, DVOUT	-
ONTDC	SI, DR	-	ONTDC	-
CONCA	FCT	N, SI, L, M	CONCA, TMA, TMB, P, OUTPUT, CC, Q	P, CC, OTN
FUNCA	FCT, P, Q	L, TMB, TBL	FUNCA, OUTPUT, F	F
FACEX	TBL	-	TMA, FACEX, L, N, TMB	RNO
ADSUB	FCT	-	ADSUB, Q, P, TMA	-
RESET	OPER	-	RESET, M, P, L	DR, C

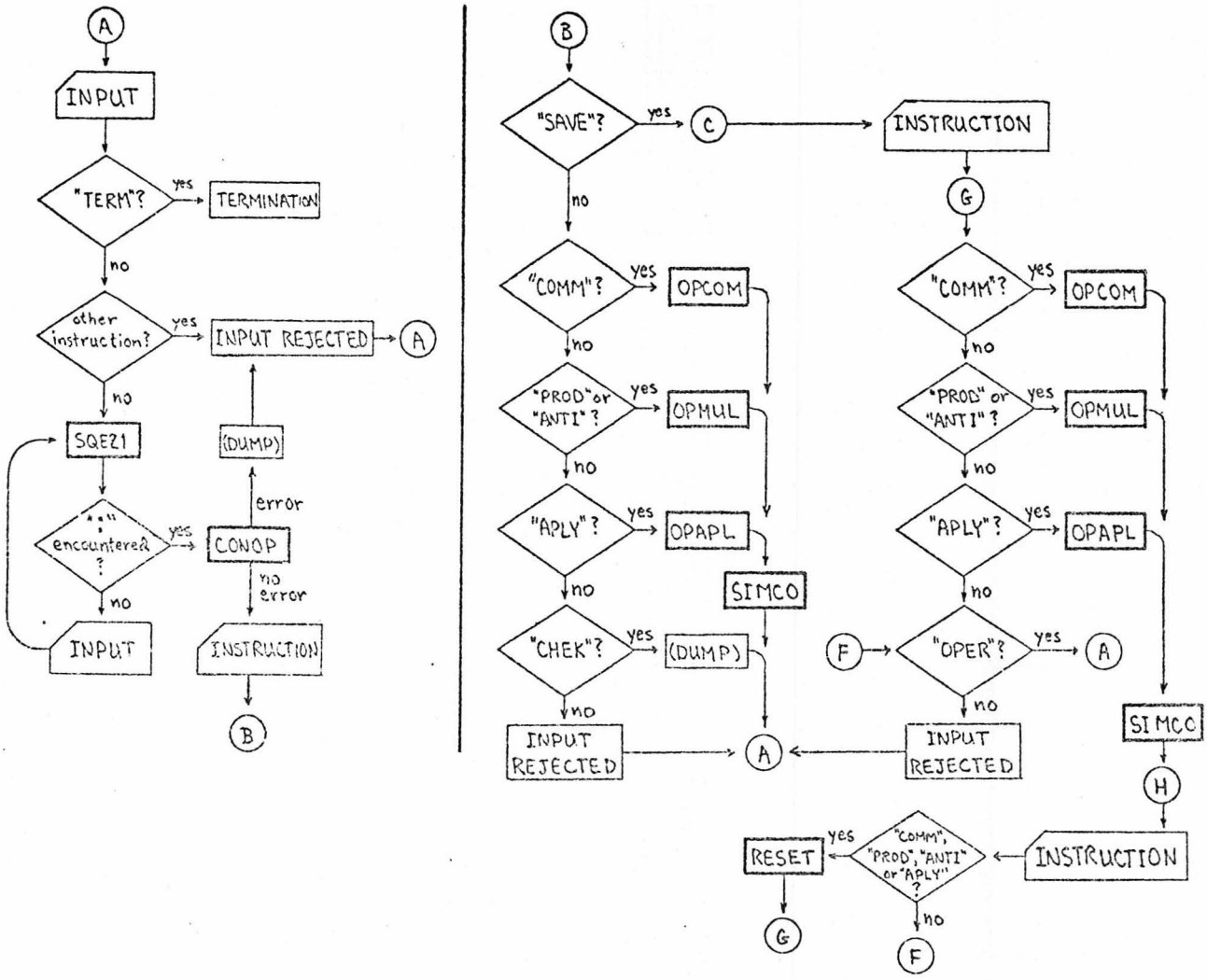

Figure 5.3.3 Overall flowchart for the program MANDO. The boxes are either p-functions (outlined with heavy lines) or other sequences. The card-shaped symbols are inputs; the diamonds, analyses; and the labelled arrows, decisions. The unlabelled arrows show mandatory flow. Positions A and B of Table 3.1, p. 15, correspond to positions A and B of this chart; position C of Table 3.1 is either position C or H of this chart. Input failure in the interactive mode also results in a return to position A (not shown); there is no counterpart for this in the batch mode.

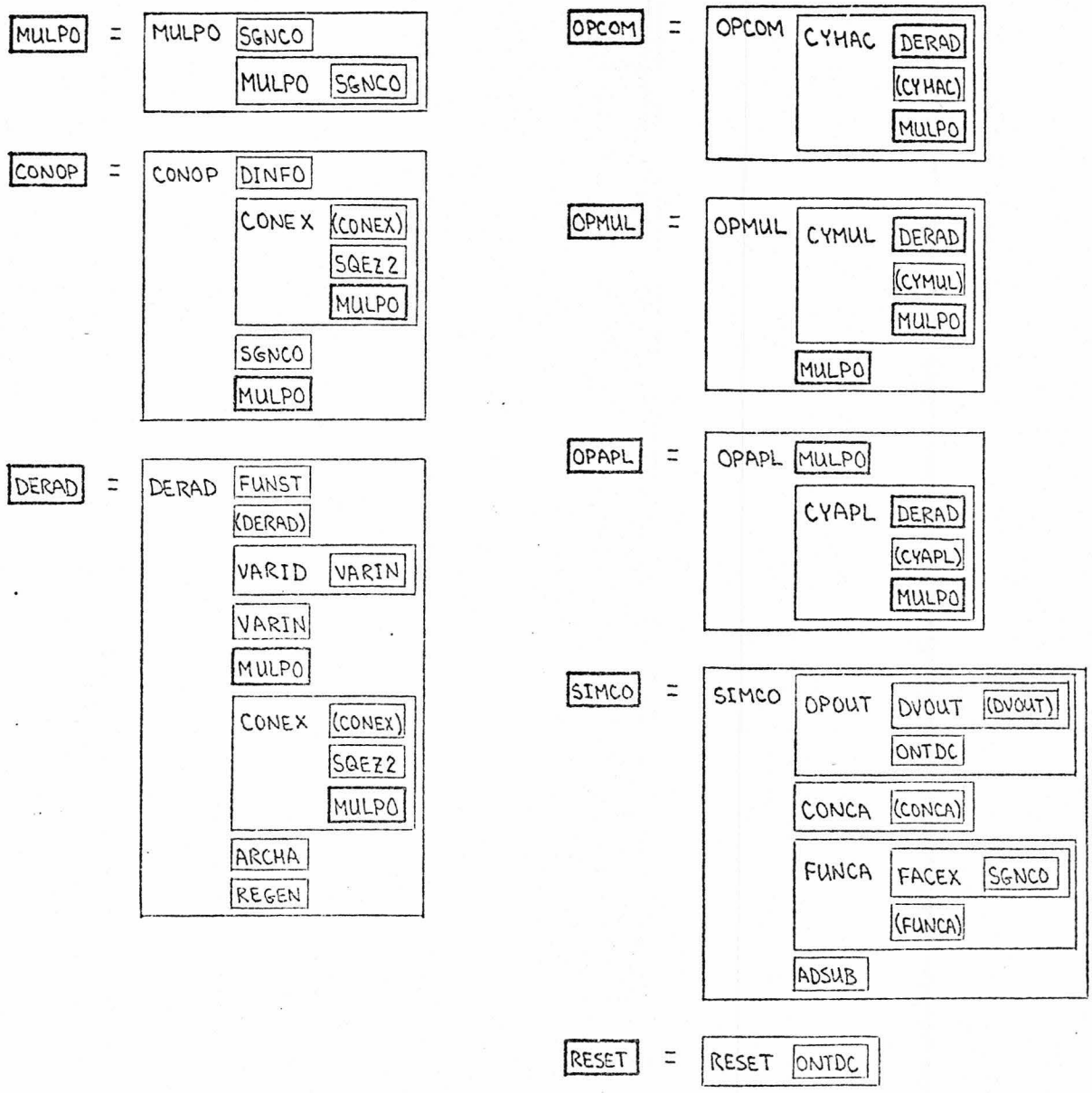

Figure 5.3.4 For brevity p-functions CONOP, OPCOM, OPMUL, OPAPL, SIMCO and RESET are indicated in Figure 5.3.3 by boxes with heavy lines. The internal structures of these and of p-functions MULPO and DERAD, abbreviated in them, are given above. These diagrams show which p-functions are called inside other p-functions.

Chapter VI

Extensions and Changes Contemplated

Several changes to MANDO suggest themselves. Each, however, requires the lengthening of the program, at the expense, during execution, of the space used for dynamical storage. In Stanford batch class B, MANDO requires 116,000 bytes to compile, leaving 91,000 at the disposal of the program during execution. More space is available in other partitions (classes O and L), but the costs of using them are from 40 to 100% per minute greater than in class B with execution priority I (IDLE), normally used. This is all to point out that it may not be desirable to extend MANDO extensively. Sections 6.1 and 6.2, however, offer some promising candidates for simple extensions to the program.

Section 6.3 describes how to modify the job control language for use with problems which take more than a minute to run or which require more space than that provided in class B.

In Section 6.4 is a discussion of the means of improving the simplification routine and of related matters.

6.1 Additional Instructions for Use with Sets of Operators

When it is desired to determine the structure of a set of n operators, the $n^2 - n$ commutators of every operator with every other operator of the set must be known. To perform these commutations, the program presently must be given each pair of operators in the for-

mat prescribed: OPERATOR "#" OPERATOR ":" (see Examples 1 and 2, Chapter IV, pp. 20 to 48). Each operator is converted to datatype OPDER, and the processing begins. After the result is returned, information about the original operators is lost. If the program were able to accept a set of operators, convert them all and store them, the commutations could be performed among the converted operators, thus saving the time taken up by conversion. It should be noted that, in the current version, if one of the operators does not convert (CONOP fails), a system dump results. Writing an operator incorrectly several times results in just as many dumps*.

If, at the beginning of the input sequence (point A on Table 3.1, p. 15, and on Fig. 5.3.3, p. 122), another option besides an operator pair or the instruction "TERM" were given, say, an instruction "CSET", the program could handle a set of operators as described above. A possible format for input of a set of operators is: OPERATOR "#" OPERATOR "#" ... "#" OPERATOR ":". Each operator could be stored as an element of an array or table OP, and the commutations would be controlled by increments of the indices of OP.

Another useful feature would be the ability to automatically determine all the commutators of operators, each from disjoint subsets of a set of operators. Thus an instruction "CROS" might cause

* Each dump produces hundreds of lines of output. Some control may be exercised, however, by limiting the number of dumps via the D parameter described on p. 11-4 of the SPITBOL Manual (4).

such a sequence of events, where the operator sets OPERATOR "#" OPERATOR "#" ... "#" OPERATOR are separated by "##" and end with ":".

If additional instructions are added to the given set, care must be exercised in their selection. The ones in MANDO and the ones above do not fit into the format for functions given in Chapter III. This is a useful feature in that, if one of the instructions is accidentally concatenated with an operator due to an error in setting up a problem for MANDO, an error will be evident on the printout. Conversely, there is no danger of a function being interpreted as an instruction when these precautions are taken.

6.2 Addition of Other Useful Functions

The program can be modified to accept functions other than exp, sin, cos, lgn and sqt. The pattern USF contains the names of these functions. Other names can be added. If the function's derivative also contains an index or indices simply related to the initial function's index, then this information should also be included in the argument. This requires the addition of a symbol, say, ",", to separate the arguments, which, in turn, requires the modification of patterns NVE, NCN and SVR to include the symbol in their SPAN sub-patterns.

To include a new function, line 748 of p-function DERAD must also be changed to include the function's name and an appropriate procedure written, following the models of those of the other "useful functions" (see lines 752 to 766).

Other precautions to be taken are to begin the new function with one of the letters "E", "S", "C" or "L", to restrict the length to three characters, and to use neither the letter "D" nor any of the variables among the last two letters of the function name.

If, for example, it is determined that the operators are best expressed in terms of Bessel functions $J_n(z)$, then a function with the name "SBE" may be used, and a typical occurrence would be "SBE(Z, N)", the Bessel coefficient of argument Z and order N. The derivative of this is "0.5SBE(Z,N-1)-0.5SBE(Z,N+1)", which can be easily written in datatype EXFUN. The difference between the derivative procedure for this function and those for the given "useful functions" is that the argument here must be taken apart, analyzed, and reassembled. In the other routines the argument is simply converted via CONEX.

The procedure above may also be used in introducing implicit functions for use in operator manipulations. Let us assume that functions "LQ0", "LQ1", "LQ2" and "LFN" are introduced and equipped with a procedure which writes the derivative of "LQ0(X,Y,...)", "LQ1(X,Y,...)", "LQ2(X,Y,...)" and "LFN(X,Y,...)", with respect to any of the variables V appearing as an argument, as "LQ0(X,Y,...,V)", "LQ1(X,Y,...,V)", "LQ2(X,Y,...,V)" and "LFN(X,Y,...,V)"; then these functions can represent implicit functions of the variables x, y, ... If, for example, on input there is an expression containing "LQ1(X,Z)", then, on output, "LQ1(X,Z,X,X)" represents the second derivative of

"LQ1(X,Z)" with respect to the variable x.

With the ability to manipulate implicit functions, program MANDO becomes able to perform the initial computations involved in arriving at a set of determining equations for a differential equation and operators of the implicit form chosen*. If, for instance, in the case of the free particle in quantum mechanics, the differential equation can be written $f_{xx} + 2if_t = 0$ and if an operator of the form $q^o + q^x\partial_x + q^t\partial_t$ is chosen, where q^o, q^x, q^t and f correspond, respectively, to the implicit functions "LQ0(X,T)", "LQ1(X,T)", "LQ2(X,T)" and "LFN(X,T)", then the determining equations would come out of an evaluation of the result of the following set of inputs in MANDO:

```
"DXDX+2IDT # LQ0(X,T)+LQ1(X,T)DX+LQ2(X,T)DT:"
"SAVE"
"PROD"
"APLY LFN(X,T)"
"OPER"
"TERM"
```

Such a result would be, in general, difficult to evaluate, however, due to the fact that MANDO does not factor out common terms within results. By modifying the program or by simply modifying the output**

* See, for example, the derivation of the determining equations 6.12 to 6.15 in Anderson et al. (7), pp. 10 to 12.

** As suggested in Section 6.4, p. 134.

and using another algebraic simplification system* to order the terms, the problem could be transformed to a factored result from which it would be easy to directly pick out the determining equations.

Using an enlarged MANDO to evaluate determining equations is wasteful in some respects. A smaller program using a more compact notation would be more efficient, though it would, of course, take longer to develop.

6.3 Job Control Language**

When the program runs in the batch mode, certain coded statements must be placed before the start of the program to make the SPITBOL system available and to set the parameters of execution. For the normal use of the program MANDO, the following job control language is used (see also lines 1 to 6, Appendix B):

```
//MANDO JOB 'C010,460,,4,','DEPT.OF.PHYSICS.UOP'
/* SERVICE EXEC=I
//  EXEC PGM=SPITBOL,PARM='R=15K,T=55,P=50,C=0'
//SYSPRINT DD SYSOUT=A
//SYSPUNCH DD SYSOUT=B
//SYSIN DD *
```

* REDUCE (10) has a declaration FACTOR which factors out all powers of its arguments. Other systems undoubtably offer similar options.

** See the Stanford Computation Center's Users Manual (11), Chapter 3, for a complete description of the job control language used here. The SPITBOL Manual (4) also gives a similar setup in Chapter 11.

The changes from the default parameters are underlined. The first allows for 4000 lines of output. The second sets the execution priority to IDLE so that a special low rate applies. The R parameter controls the amount of space used for input and is necessary. The time limit is controlled by the parameter T (set at 55 seconds: if T is set much higher than this a job terminating by exceeding the time limit will not include a dump in its output). P controls the number of pages of output (50 pages is about 3000 lines). Setting C to zero indicates that no cards will be punched.

When it appears that the program will be required to run more than a minute, three changes must be made to the job control statements: (1) in the space to the right of the "460," (bin number for delivery of printouts) in the first line, an outside time estimate must be entered (in minutes and tenths of minutes); (2) the execution priority must be changed; and (3) the time parameter in the third line must be changed. Since programs which run more than a minute cannot be run with execution priority I (IDLE) and since they cost 80% per minute more under the next execution priority of class B, it is generally wise to run instead in batch class O, which costs 40% per minute more but provides more dynamic memory and thus shorter execution times. For an anticipated run of three minutes in batch class O, the first three lines of the job control language statements are changed to:

```
//MANDO JOB 'C010,460,3,10,','DEPT.OF.PHYSICS.UOP'
/* SERVICE CLASS=O
//  EXEC PGM=SPITBOL,PARM='R=15K,T=175,P=120,C=0'
```

The limit on the number of pages and lines allowed is also increased to allow for the inclusion of all the results and a dump if failure occurs. In addition to these changes, &STLIMIT should be changed to 600,000 (200,000 per minute is a safe overestimate - see p. 97).

Even though a job is expected to run less than a minute, batch class O may cost less than class B, priority I, especially when the job may take more than 10 seconds or so to run. If the program is used extensively in one class, it is advisable to run an identical problem in the other once in a while to compare their costs. The execution statistics at the end of each run include the timing; the RALPH log, from which the charges are computed, also appears on the printout.

One reason identical problems take less time in class O is that fewer storage regenerations are required. Some time loss is expected when working with a larger memory space due to increased access and storage regeneration time, but these effects appear to be insignificant here. From an examination of the printouts of identical problems for which the amount of dynamic memory available was slightly different and therefore the numbers of storage regenerations different, the author concludes that storage regenerations take about a twentieth of a second in batch class B.

The advantage that class B has over class O is that it is available during the day, though jobs with priority I sometimes take several hours to be run.

6.4 Improvement of Simplification Routine

For Example 4, Chapter IV, the "simplified result" (pp. 65 and 66) is considerably more readable than the "ordered result" (pp. 61 to 65), but there is still room for improvement. The factorization performed in the margin of the printout of that example, performed to clarify and further reduce the results, suggests, however, the manner in which one would proceed to set up an automatic routine for further simplification. The lowest power of each factor is determined. This is extracted as the factor common to all terms, and the terms are adjusted to reflect this extraction. The result is a factor times a polynomial, the latter of which can be further simplified, as was previously done by hand. As regards the capacity of the EXFUN datatype to store information about functions, a problem arises in that the polynomial factor generally has more terms than the original function, e. g., "Y(1-X@(2))@(2)-X@(4)Y" factors like "Y(1-2X@(2)+X@(4)-X@(4)". This can be remedied by using tables in the EA fields or by using p-function ARCHA to create the necessary space. The common multiplier can be stored in a third field of a newly defined datatype EXFUN: DATA('EXFUN(EI,EA,EF)'). By writing the EF field as a string, recognition is made of the fact that the common factor applies to a particular field only.

The above procedure, outlined for the case of Example 4, Chapter IV, can be generalized to other cases also. The distinction can be made between this type of factorization in the name of simplification and another type, which results in the ordering of terms

around powers of certain variables. The latter type would be of use in the solution of problems such as that presented on p. 128., where a means of expressing functions in terms of embedded multiplicative factors is required. Such a routine would rewrite, say, $3x^2y^2 - 2x^2yz^3 + 5x^2z^2 + 4x - 6y^3z + y^3 + 3y^2 + z^4 + 1$ as $(3y^2 - (2z^3)y + 5z^3)x^2 + (4)x + ((-6z + 1)y^3 + 3y^2 + z^4 + 1)$.

An alternative to writing a string-manipulation sequence for the above evaluation would be to write a list-processing routine*. This would require, at the least, the conversion of the AC and AT fields to lists (easily in modifications of p-functions CONCA and FUNCA). Using lists here, however, requires the addition of several list-processing sequences which would undoubtably slow down the program, large as it is already. A major restructuring of the entire program, replacing its current data structures with branched lists, would be a more fruitful means of capitalizing on the past work which has shown the list-processing language LISP (13) to be popular for writing programs for symbolic algebraic manipulation. But this would be tantamount to yielding that the program would be better written in LISP in the first place and to throwing away the conceivable advantages of the current version, plus a string-processing simplification and factorization extension, might have. As regards space, the room taken up by a string is undoubtably less than that taken up by its

* In section 9 of (12), Wegner discusses the data structure of SNOBOL4. Included is a description of how the list-processing language LISP can be simulated by SNOBOL. See also Section 1.14 of (5).

equivalent list structure. The other consideration, timing, awaits the development of the extension to MANDO and the concomitant modification of current symbolic algebra systems to be able to directly perform operations such as those of which MANDO is capable.

An alternative to improvements of the simplification sequence in MANDO is the slight modification to functions CONCA and FUNCA to cause them to write AC and AT fields in FORTRAN format and the equivalently minor changes to OPOUT to complete the creation of an overall output in FORTRAN format. The operator, in this form, could be further manipulated by an existing algebraic manipulation system. The sets of derivative coefficients (such as "DRDXDX"), all unique, could be interpreted by such systems as variables and could be used as such in an ordering and factorization routine.

Another use for FORTRAN format on output is the determination of the change in value of a function which has undergone a transformation under the action of some operator (via evaluation of the terms in the expansion to some order, as with Example 3, Chapter IV, pp. 49 to 60).

REFERENCES

1. Sophus Lie, Vorlesungen ueber Differentialgleichungen mit Bekannten Infinitesimalen Transformationen, bearbeitet und heraus gegeben von Dr. Georg Scheffers, Leipzig, B. G. Teubner, 1891.

2. Abraham Cohen, An Introduction to the Lie Theory of One-parameter Groups, D. C. Heath & Co., New York, 1911.

3. R. L. Anderson, S. Kumei, and C. E. Wulfman, "Generalization of the Concept of Invariance of Differential Equations. Results of Applications to Some Schroedinger Equations", Phys. Rev. Lett.,

4. R. B. K. Dewar and K. Belcher, SPITBOL Manual - Version 2.0, Stanford Computation Center, Stanford University, no publication date given.

5. R. E. Griswold, J. F. Poage, and I. P. Polonsky, The SNOBOL4 Programming Language, 2nd Ed., Prentice-Hall, Englewood Cliffs, NJ, 1971.

6. J. E. Campbell, Introductory Treatise on Lie's Theory of Finite Continuous Transformation Groups, Chelsea, Bronx, NY, 1903 (reprint: 1966).

7. R. L. Anderson, S. Kumei, and C. E. Wulfman, "Invariants of the Equations of Wave Mechanics I" and "Invariants of the Equations of Wave Mechanics II - One Particle Schroedinger Equations", Rev. Mex. Fis., v. 21, p. 1 and p. 35 (1972).

8. J. Earley, "Towards an Understanding of Data Structures", Comm. ACM 14, 10 (Oct 71), p. 617.

9. J. Moses, "Algebraic Simplification: A Guide for the Perplexed", Comm. ACM 14, 8 (Aug 71), p. 527.

10. A. C. Hearn, REDUCE2 User's Manual, Computer Science Department Report STAN-CS-70-181, Stanford University, Oct 70.

REFERENCES
(continued)

11. Users Manual, Computation Center, Stanford University, 1972.

12. P. Wegner, "Data Structure Models for Programming Languages", SIGPLAN Notices 6, 2 (Feb 71), p. 1.

13. C. Weissman, LISP 1.5 Primer, Dickenson Pub. Co., Inc., Belmont, Calif., 1967.

Appendix A

Justification of Format Chosen

Many algebraic manipulation systems retain the FORTRAN-inspired "*" to represent multiplication. There are two reasons for this. The first is that, when any letter or group of letters and symbols, e. g., "X", "X1", "RHO" and "SIGMA", may be used as a factor in a product, it must be set off by asterisks to distinguish it; otherwise an expression like "XYESZ" could mean "X*YES*Z" or "X*Y*E*S*Z", a distinction which must be worked out by a declaration of the variables used. The second reason for the inclusion of the "*" in the format is to ensure that a calculated result be readable† by a FORTRAN-like compiler in case it is desired that the same evaluate the resulting expression for some values of the variables. This procedure would be adopted for a problem which would otherwise require more time by outright numerical evaluation.

At the present stage work with differential operators requires the use of very few variables (variables in the mathematical sense; in the computer sense, many are used in this program) and only a few constants. Since variables and constants are formally restricted (essentially declared) in the program, since the results

† A statement such as $r = u^2(\sin^2 3gx)v$ would be written as "R=U**2 *(SIN(3*G*X))**2*V". Results written in this form are in FORTRAN format.

are not evaluated for any values of the variables, and since the results of operator calculations are, in general, not compatible with FORTRAN format anyway (What do you do with the derivative coefficients?), the format for the program MANDO does not use "*" to indicate multiplication†.

Exponentiation is represented by "@"‡. "**" and "!" are the two standard symbols. The first has the disadvantage of requiring two spaces; the second looks too much like "1" and "I" on the printouts.

Another feature of the format chosen is its close similarity to the format stored within the datatypes in the computer during execution. Conversion from string to datatype is done by pattern matching, and the reverse, by concatenation (with appropriate modification). In other systems functions are often stored in Polish notation: 2x - 6 is (- (* 2 x) 6). It is likely that conversion for them to a machine form takes more time than for MANDO, especially where fairly complex functions are concerned.

† Such a format may, however, be useful in some cases - see Section 6.4, p. 134.

‡ To the author's knowledge, this symbol was first used in a SNOBOL derivative program, &p799.math.deriv on sys08 in the Stanford library of programs, written by R. Code of the Computation Center.

```
                        MANDO    2 AUG 73
 1.      //MANDO JOB 'CC10,460,,4,',DEPT.OF.PHYSICS,UCP'
 2.      /* SERVICE EXEC=I
 3.      //  EXEC PGM=SPITBOL,PARM='R=15K,T=55,P=50,C=0'
 4.      //SYSPRINT DD SYSOUT=A
 5.      //SYSPUNCH DD SYSOUT=B
 6.      //SYSIN DD *
 7.      ********************************************************************************
 8.      **     PROGRAM FOR SYMBOLIC MANIPULATION OF DIFFERENTIAL OPERATORS      **
 9.      ********************************************************************************
10.      -UNLIST
11.      -STAT
12.      *  KEYWORDS AND TRACE OPERATIONS                                          **
13.            &STLIMIT = 200000  ;  &DUMP = 2
14.            &ANCHOR = 1  ;  &TRIM = 1
15.      *  DATATYPES                                                              **
16.            DATA('OPDER(OI,OA)')
17.            DATA('EXFUN(EI,EA)')
18.            DATA('ADDEX(AS,AN,AC,AT,AD)')
19.      *  FUNCTIONS                                                              **
20.            DEFINE('SQEZ1(OPER)')
21.            DEFINE('CONOP(OPER)TBL')
22.            DEFINE('DINFO(DR)')
23.            DEFINE('CONEX(FCT)AS,AN,AC,AT,AD,P,SI')
24.            DEFINE('SQEZ2(FCT)P,Q,TMA,ARG')
25.            DEFINE('MULPO(TMA,TMB,SGN)A,B')
26.            DEFINE('SGNCO(SGNCO)')
27.            DEFINE('OPCOM(OPA,OPB)A,B')
28.            DEFINE('CYHAC(OPA,OPB,A,B,VAR,VR2,VR3,VR4,VR5,VR6,VR7,VR8,VR9)'
29.      .     'DR,TMA,TMB')
30.            DEFINE('OPMUL(OPA,OPB,SI)A,B')
31.            DEFINE('CYMUL(OPA,OPB,A,B,VAR,VR2,VR3,VR4,VR5,VR6,VR7,VR8,VR9)'
32.      .     'DR,TMA,TMB')
33.            DEFINE('OPAPL(OPA,FCT)')
34.            DEFINE('CYAPL(OPA,FCT,VAR,VR2,VR3,VR4,VR5,VR6,VR7,VR8,VR9)DR')
35.            DEFINE('DERAD(FCT)DR,ARG')
36.            DEFINE('FUNST(FCT)P')
37.            DEFINE('ARCHA(TMA,TMB,SI)P,Q')
38.            DEFINE('VARID(FCT)')
39.            DEFINE('VARIN(FCT)')
40.            DEFINE('REGEN(TBL)')
41.            DEFINE('SIMCO(OPER)')
42.            DEFINE('OPOUT(OPER)')
43.            DEFINE('DVOUT(FCT)P')
44.            DEFINE('ONTOC(SI,DR)')
45.            DEFINE('CONCA(FCT)N,SI,L,M')
46.            DEFINE('FUNCA(FCT,P,Q)L,TMB,TBL')
47.            DEFINE('FACEX(TBL)')
48.            DEFINE('ADSUB(FCT)')
49.            DEFINE('RESET(OPER)')
50.      *  REPEATED PATTERNS                                                      **
51.            ANY = ANY('RTUVWXYZ')  ;  RNO = SPAN('1234567890.')
52.            NVE = SPAN('|#+-/@()ABDFGHIJKMNOPQ1234567890.')
53.            NOV = ((NVE | NULL) ANY('ESCL') LEN(2) NVE *NOV | NULL)
54.            CNP = SPAN('ABFGHIJKMNOPQ12345')
55.            CNA = ANY('+-') (RNO (CNP | NULL) | CNP) (*CNA | NULL)
56.            CNS = (((CNP | NULL) '(' (ANY('+-') | NULL) (RNO (CNP |
57.      .     NULL) | CNP) CNA ')' | CNP '@(' RNO ')') (*CNS | NULL) | CNP)
58.            NCN = SPAN('|#+-/@()DIRTUVWXYZ1234567890.')
59.            NOC = ((NCN | NULL) ANY('ESCL') LEN(2) NCN *NOC | NULL)
```

```
60.             ACN = ANY('ABFGHJKMNOPQ') (SPAN('12345') | NULL)
61.             SCO = (SPAN(')ABFGHIJKMNOPQ1234567890.') '@(' *SCO | NULL)
62.             USF = ('EXP(' | 'SIN(' | 'COS(' | 'LGN(' | 'SQT(')
63.             SVR = ((SPAN('+-/)ABFGHIJKMNOPQ1234567890.RTUVWXYZ') | NULL)
64.     .       (('(' BAL ')' | NULL) '@(' | USF) BAL ')' *SVR | NULL)
65.             SFU = (((ANY('ESCL') LEN(3) BAL ')' | SPAN('RTUVWXYZ')) ('@('
66.     .       BAL ')' | NULL) | '(' BAL ')' ('@' FAIL | NULL)) *SFU | NULL)
67.             OTN = ANY('012345678')
68.             FAD = EXFUN(1,ADDEX('+',1,,'$$',))
69.     *   INITIAL ASSIGNMENTS AND CREATION OF ARRAY AND TABLES                **
70.             CMP = TABLE(10)  ;  CMP<0> = ARRAY(4)  ;  CMP<0><2> = 10
71.             CMP<0><3> = 30  ;  CMP<0><4> = 40  ;  CMP<11> = TABLE(4,2)
72.             CCF = TABLE(20,50)  ;  FCF = TABLE(30,80)  ;  CC = 1  ;  F = 1
73.             PCT = TABLE(10)  ;  PFT = TABLE(30,30)
74.             J = 1  ;  K = 29  ;  DTA = TABLE(20,20)  ;  TDT = TABLE(30)
75.             S = 1  ;  IN = TABLE(3,5)  :(PROST)
76.     *   INPUT SEQUENCE                                                     **
77.     REJIN   OUTPUT = ' INPUT REJECTED.'  ;  OUTPUT =  ;  OUTPUT =
78.     PROST   OUTPUT = DUPL('* -- *',20)  ;  OUTPUT =
79.             OP = ARRAY('2,2')  ;  O = 1
80.             OUTPUT = '*$$* BEGINNING OF INPUT SEQUENCE, TIME:  ' TIME()
81.     .       ' MSEC, COUNT:  ' &STCOUNT ' STATEMENTS *$$*'
82.             OUTPUT = ' OPERATOR PAIR, SEPARATED WITH A "#" AND ENDING WITH '
83.     .       'A COLON ":" TO START PROCESSING, OR "TERM" TO TERMINATE:'
84.     INOUT   INPUT (SPAN(' ') | NULL) REM $ OUTPUT  :F(REJIN)
85.             IDENT(OUTPUT,'TERM')  :S(TERSQ)  ;  OUTPUT ('COMM' | 'ANTI' |
86.     .       'PROD' | 'CHEK' | 'SAVE' | 'APLY' | 'OPER')  :S(REJIN)
87.             OPER = SQEZ1(OUTPUT)
88.     *     ASSIGNMENT OF OPERATORS TO ARRAY ELEMENTS                        *
89.     PEELO   OPER BREAK('#') . TMA '#' =  :S(OVINC)
90.             OPER BREAK(':') $ OPER  :F(AWHOP)
91.             OP<O,2> = OP<O,2> OPER  :(TWOSQ)
92.     AWHOP   OP<O,2> = OP<O,2> OPER  :(INOUT)
93.     OVINC   OP<O,2> = OP<O,2> TMA  ;  O = LT(O,2) O + 1  :S(PEELO)
94.             OUTPUT = ' INCORRECT FORM FOR INPUT OF OPERATOR PAIR (WRITE:  '
95.     .       'OPERATOR "#" OPERATOR ":").'  :(REJIN)
96.     *     CONVERSION AND REASSIGNMENT OF OPERATORS                         *
97.     TWOSQ   O = 1
98.     ASGOC   OUTPUT = IDENT(OP<O,2>) ' NULL OPERATOR (' O ')'  :S(REJIN)
99.             OP<O,2> BREAK('|') . OP<O,1> '|(' RTAB(1) . OP<O,2> ')'
100.    .       :F(NOLMU)
101.            OP<O,1> = CONEX(OP<O,1>)
102.    NOLMU   OP<O,2> = CONOP(OP<O,2>)  :F(FAICO)
103.    INCIP   O = LT(O,2) 2  :S(ASGOC)F(COSEQ)
104.    FAICO   O = LT(O,2) 2  :F(DMPAR)
105.            OUTPUT = IDENT(OP<2,2>) ' NULL OPERATOR (2)'  :S(DMPAR)
106.            OP<2,2> BREAK('|') . OP<2,1> '|(' RTAB(1) . OP<2,2> ')'
107.    .       :F(LMUNO)
108.            OP<2,1> = CONEX(OP<2,1>)
109.    LMUNO   OP<2,2> = CONOP(OP<2,2>)
110.    DMPAR   OUTPUT =  ;  OUTPUT = ' DIAGNOSTIC DUMP DUE TO CONVERSION '
111.    .       '(CONOP) FAILURE:'  ;  OUTPUT =  ;  DUMP(2)  :(REJIN)
112.    *     DETERMINATION OF OPERATION TO BE PERFORMED                       *
113.    COSEQ   OUTPUT = ' "SAVE" (FOLLOWED BY "COMM", "ANTI", "PROD" OR '
114.    .       '"APLY" AS NEXT INPUT), "COMM", "ANTI", "PROD", "APLY" OR '
115.    .       '"CHEK":'  ;  OUTPUT = ' (TIME,COUNT):  (' TIME() ' MSEC, '
116.    .       &STCOUNT ' STATEMENTS)'
117.            INPUT (SPAN(' ') | NULL) REM $ OUTPUT  :F(REJIN)
118.            OUTPUT = IDENT(OUTPUT,'SAVE')  :F(NOSAV)
119.    *     SEQUENCE FOR CYCLIC OPERATIONS                                   *
```

```
120.                OUTPUT = DUPL('* -- *',20)  ;  OUTPUT =
121.                OUTPUT = '*$$* ENTRY INTO EXECUTION CYCLE *$$*'
122.                OUTPUT = ' "COMM", "ANTI", "PROD" OR "APLY":'
123.        IOTWO   INPUT (SPAN(' ') | NULL) REM $ OUTPUT  :F(REJIN)
124.        IDOCM   OP<2,2> = IDENT(OUTPUT,'COMM') OPCOM(OP<1,2>,OP<2,2>)
125.        .       :F(ANOPC)
126.                OUTPUT = ' TIME AND COUNT AT END OF COMMUTATION AND BEGINNING '
127.        .       'OF SIMPLIFICATION FUNCTION (SIMCO):  ' TIME() ' MSEC, '
128.        .       &STCOUNT ' STATEMENTS.'  :(RESMP)
129.        ANOPC   OP<2,2> = IDENT(OUTPUT,'ANTI') OPMUL(OP<1,2>,OP<2,2>,2)
130.        .       :F(PTWOP)
131.                OUTPUT = ' TIME AND COUNT AT END OF ANTICOMMUTATION AND '
132.        .       'BEGINNING OF SIMPLIFICATION (SIMCO):  ' TIME() ' MSEC, '
133.        .       &STCOUNT ' STATEMENTS.'  :(RESMP)
134.        PTWOP   OP<2,2> = IDENT(OUTPUT,'PROD') OPMUL(OP<1,2>,OP<2,2>,1)
135.        .       :F(AFUCY)
136.                OUTPUT = ' TIME AND COUNT AT END OF PRODUCT ROUTINE AND '
137.        .       'BEGINNING OF SIMPLIFICATION (SIMCO):  ' TIME() ' MSEC, '
138.        .       &STCOUNT ' STATEMENTS.'
139.        RESMP   OP<2,2> = SIMCO(OP<2,2>)  :S(PILMT)
140.                CCF = GT(CC,50) TABLE(20,50)  :F(FUTA1)  ;  CC = 1
141.        FUTA1   FCF = GT(F,80) TABLE(30,80)  :F(NULO1)  ;  F = 1
142.        NULO1   OUTPUT =  ;  OUTPUT = ' NULL RESULT.'  ;  OUTPUT =
143.                OUTPUT =  ;  OUTPUT =  ;  OUTPUT = DUPL('* -- *',20)  :(PROST)
144.        PILMT   OP<2,1> = MULPO(OP<1,1>,OP<2,1>,)
145.                OUTPUT =  ;  OUTPUT = '*$$* CONTINUATION OF EXECUTION CYCLE, '
146.        .       'TIME:  ' TIME() ' MSEC, COUNT:  ' &STCOUNT ' STATEMENTS *$$*'
147.                OUTPUT = ' "COMM", "ANTI", "PROD", "APLY" OR "OPER":'
148.                INPUT (SPAN(' ') | NULL) REM $ OUTPUT  :F(REJIN)
149.                OUTPUT 'COMM' | 'ANTI' | 'PROD' | 'APLY'  :F(RETFA)
150.                OP<2,2> = RESET(OP<2,2>)  :(IDOCM)
151.        AFUCY   OUTPUT 'APLY' REM $ FCT  :F(RETFA)
152.                FCT = OPAPL(OP<2,2>,CONSX(SQEZ1(FCT)))
153.                OUTPUT = ' TIME AND COUNT AT END OF APPLICATION ROUTINE AND '
154.        .       'BEGINNING OF SIMPLIFICATION (SIMCO):  ' TIME() ' MSEC, '
155.        .       &STCOUNT ' STATEMENTS.'  ;  SIMCO(FCT)  :S(INFOC)
156.                CCF = GT(CC,50) TABLE(20,50)  :F(FUTA2)  ;  CC = 1
157.        FUTA2   FCF = GT(F,80) TABLE(30,80)  :F(NULO2)  ;  F = 1
158.        NULO2   OUTPUT =  ;  OUTPUT = ' NULL RESULT.'  ;  OUTPUT =
159.                OUTPUT =  ;  OUTPUT =  ;  OUTPUT = DUPL('* -- *',20)
160.        INFOC   OUTPUT =  ;  OUTPUT = '*$$* CONTINUATION OF EXECUTION CYCLE, '
161.        .       'TIME:  ' TIME() ' MSEC, COUNT:  ' &STCOUNT ' STATEMENTS *$$*'
162.                OUTPUT = ' "COMM", "ANTI", "PROD", "APLY" OR "OPER":'
163.                INPUT (SPAN(' ') | NULL) REM $ OUTPUT  :F(REJIN)
164.                OUTPUT 'COMM' | 'ANTI' | 'PROD' | 'APLY'  :S(IDOCM)
165.        RETFA   OUTPUT = DIFFER(OUTPUT,'OPER') ' UACCEPTABLE EXIT FROM TWO-'
166.        .       'OPERATOR ALGEBRA ROUTINE (USE "OPER")'  :S(REJIN)
167.                OUTPUT =  :(PROST)
168.        *    SEQUENCE FOR ONE-TIME OPERATION                                *
169.        NOSAV   OP<2,2> = IDENT(OUTPUT,'COMM') OPCOM(OP<1,2>,OP<2,2>)
170.        .       :F(ANTIC)
171.                OUTPUT = ' TIME AND COUNT AT END OF COMMUTATION AND BEGINNING '
172.        .       'OF SIMPLIFICATION FUNCTION (SIMCO):  ' TIME() ' MSEC, '
173.        .       &STCOUNT ' STATEMENTS.'  :(SIRSL)
174.        ANTIC   OP<2,2> = IDENT(OUTPUT,'ANTI') OPMUL(OP<1,2>,OP<2,2>,2)
175.        .       :F(PRTWO)
176.                OUTPUT = ' TIME AND COUNT AT END OF ANTICOMMUTATION AND '
177.        .       'BEGINNING OF SIMPLIFICATION (SIMCO):  ' TIME() ' MSEC, '
178.        .       &STCOUNT ' STATEMENTS.'  :(SIRSL)
179.        PRTWO   OP<2,2> = IDENT(OUTPUT,'PROD') OPMUL(OP<1,2>,OP<2,2>,1)
```

```
180.      .       :F(APLYO)
181.              OUTPUT = ' TIME AND COUNT AT END OF PRODUCT ROUTINE AND '
182.      .       'BEGINNING OF SIMPLIFICATION (SIMCO):  ' TIME() ' MSEC, '
183.      .       &STCOUNT ' STATEMENTS.'
184.      SIFSL   SIMCO(OP<2,2>)  :S(PROST)
185.              CCF = GT(CC,50) TABLE(20,50)  :F(FUTA3)  ;  CC = 1
186.      FUTA3   FCF = GT(F,80) TABLE(30,80)  :F(NULO3)  ;  F = 1
187.      NULO3   OUTPUT =  ;  OUTPUT = ' NULL RESULT.'  ;  OUTPUT =
188.              OUTPUT =  ;  OUTPUT =  ;  OUTPUT = DUPL('* -- *',20)  :(PROST)
189.      APLYO   OUTPUT 'APLY' REM $ FCT  :F(CKONL)
190.              FCT = OPAPL(OP<2,2>,CONEX(SQEZ1(FCT)))
191.              OUTPUT = ' TIME AND COUNT AT END OF APPLICATION ROUTINE AND '
192.      .       'BEGINNING OF SIMPLIFICATION (SIMCO):  ' TIME() ' MSEC, '
193.      .       &STCOUNT ' STATEMENTS.'  ;  SIMCO(FCT)  :S(PROST)
194.              CCF = GT(CC,50) TABLE(20,50)  :F(FUTA4)  ;  CC = 1
195.      FUTA4   FCF = GT(F,80) TABLE(30,80)  :F(NULO4)  ;  F = 1
196.      NULO4   OUTPUT =  ;  OUTPUT = ' NULL RESULT.'  ;  OUTPUT =
197.              OUTPUT =  ;  OUTPUT =  ;  OUTPUT = DUPL('* -- *',20)  :(PROST)
198.      CKONL   OUTPUT = IDENT(OUTPUT,'CHEK') ' DIAGNOSTIC DUMP RESULTING FROM '
199.      .       '"CHEK" INPUT:'  :F(UNACI)  ;  OUTPUT =  ;  DUMP(2)  :(PROST)
200.      UNACI   OUTPUT = IDENT(OUTPUT,'OPER') ' "OPER" IS USED ONLY AFTER '
201.      .       '"SAVE".'  :(REJIN)
202.      ********************************************************************************
203.      ****  DETERMINATION AND ORDERING OF VARIABLES AND CONSTANTS  ***********
204.      SQEZ1   Q = DIFFER(OPER) 0  :F(FRETURN)  ;  &ANCHOR = 0
205.      RMBLK   OPER SPAN('* ') =  :S(RMBLK)  ;  &ANCHOR = 1  ;  SQEZ1 = OPER
206.      *  VARIABLES R,T,U,V,W,X,Y, AND Z                                        **
207.      VRSQE   OPER TAB(Q) (NVE | NOV) ANV $ TMA @Q  :S(VAIIN)
208.              Q = 0  :(CNSQE)
209.      *     CHECK OF EXISTING VARIABLE SET                                     *
210.      VAIIN   P = 1
211.      VRCOA   CMP<P> = GT(P,CMP<0><1>) TMA  :F(LAVPI)
212.              IN<TMA> = CMP<0><1>  ;  CMP<0><1> = CMP<0><1> + 1  :(VRSQE)
213.      LAVPI   P = DIFFER(CMP<P>,TMA) P + 1  :S(VRCOA)F(VRSQE)
214.      *  CONSTANTS                                                           **
215.      CNSQE   OPER TAB(Q) (NCN | NCC) ACN $ TMA @Q  :F(RETURN)
216.      *     CHECK OF EXISTING SET OF CONSTANTS                               *
217.      CNIIN   P = 41
218.      CNCOA   CMP<P> = GT(P,CMP<0><4>) TMA  :F(LACPI)
219.              CMP<0><4> = CMP<0><4> + 1  :(CNSQE)
220.      LACPI   P = DIFFER(CMP<P>,TMA) P + 1  :S(CNCOA)F(CNSQE)
221.      ********************************************************************************
222.      ****  CONVERSION TO DATATYPE OPDER  ***********************************
223.      CONOP   OPER '(' BAL . OPER ')' RPOS(0)  :S(CONOP)
224.              P = 1  ;  TBL = TABLE(4,5)
225.      *  DETERMINATION OF ADDITIVE COMPONENTS IN OPERATOR                      **
226.      ACINO   OPER BAL . TBL<P> (ANY('+-') BAL) . OPER RPOS(0)  :F(ASBEX)
227.              P = P + 1  :(ACINO)
228.      ASBEX   TBL<P> = OPER  ;  CONOP = OPDER(P,ARRAY('4,' P))  ;  P = 1
229.      *  DETERMINATION OF DERIVATIVE COEFFICIENT SETS IN COMPONENTS            **
230.      CYOPD   TBL<P> BREAK('D') . TBL<P> REM . OA(CONOP)<2,P>  :F(FUPAR)
231.      *     FORM CHECK OF DERIVATIVE COEFFICIENT SET                          *
232.              OA(CONOP)<2,P> = DINFO(OA(CONOP)<2,P>)  :F(IFDIS)
233.      *  INCLUSION OF MULTIPLIER INTO COMPONENT                               **
234.      FUPAR   OA(CONOP)<1,P> = MULPO(OP<0,1>,CONEX(TBL<P>))
235.              P = LT(P,OI(CONOP)) P + 1  :S(CYOPD)F(RETURN)
236.      *  FAILURE OF OPERATOR CONVERSION                                       **
237.      IFDIS   OUTPUT = ' INCORRECTLY FORMED DERIVATIVE COEFFICIENT SET, '
238.      .       'TENTH OR HIGHER ORDER OR FORMAT ERROR IN COMPONENT'
239.              OUTPUT = ' OF OPERATOR ' O ':  ' TBL<P> OA(CONOP)<2,P>
```

```
240.      .       :(FRETURN)
241.      ************************************************************************
242.      ****  DC SET FORM CHECK AND ASSIGNMENT  ***********************************
243.      DINFO   DR 'D' ANY $ VAR =  :F(NULDI)
244.              OA(CONOP)<4,P> = OA(CONOP)<4,P> VAR
245.              DINFO = DINFO + 10 ** IN<VAR>  :(DINFO)
246.      NULDI   OA(CONOP)<3,P> = IDENT(DR) SIZE(OA(CONOP)<4,P>)  :S(RETURN)
247.      .       F(FRETURN)
248.      ************************************************************************
249.      ****  CONVERSION FROM STRING TO EXFUN FORMAT  ****************************
250.      CONEX   FCT '(' BAL . FCT ')' RPOS(0)  :S(CONEX)
251.              CONEX = TABLE(5,10)  ;  P = 1  ;  Q = 1
252.              FCT BAL . SI (ANY('+-') BAL) . FCT RPOS(0)  :F(ONEEX)
253.      *  FIRST ADDITIVE COMPONENT                                          **
254.              SI (ANY('+-') | NULL) $ AS FENCE (RNO | NULL) $ AN FENCE
255.      .       (CNS | NULL) $ AC FENCE ((BAL | NULL) . AT '/(' BAL . AD ')'
256.      .       RPOS(0) | (REM | NULL) $ AT NULL $ AD) =  ;  AN = EQ(AN,0) 1
257.              AS = IDENT(AS) '+'  ;  AD = DIFFER(AC) CONEX(AD)  ;  SQEZ2(AT)
258.      *    EXPANSION OF PARENTHESIZED FACTORS                              *
259.      SUCN1   AC SCO . TMA '(' BAL . TMB ')' = TMA  :F(SUVR1)
260.              SI = MULPO(SI,CONEX(TMB),)  :(SUCN1)
261.      SUVR1   AT SVR . TMA '(' BAL . TMB (')@' ABORT | ')') = TMA  :F(TESI1)
262.              SI = MULPO(SI,CONEX(TMB),)  :(SUVR1)
263.      *    FIRST EXPANDED TERM                                             *
264.      TESI1   CONEX<1> = DIFFER(SI) EQ(EI(SI),1) ADDEX(SGNCO(AS(EA(SI)) AS),
265.      .       AN(EA(SI)) * AN,AC(EA(SI)) AC,AT(EA(SI)) AT,MULPO(AD(EA(SI)),AD,
266.      .       ))  :S(INVAP)
267.              CONEX<1> = DIFFER(SI) ADDEX(SGNCO(AS(EA(SI)<1>) AS),
268.      .       AN(EA(SI)<1>) * AN,AC(EA(SI)<1>) AC,AT(EA(SI)<1>) AT,
269.      .       MULPO(AD(EA(SI)<1>),AD,))  :F(CONA1)
270.      *    REMAINING EXPANDED TERMS                                        *
271.      TSEI1   P = LT(P,EI(SI)) P + 1  :F(INVAP)
272.              CONEX<P> = ADDEX(SGNCO(AS(EA(SI)<P>) AS),
273.      .       AN(EA(SI)<P>) * AN,AC(EA(SI)<P>) AC,AT(EA(SI)<P>) AT,
274.      .       MULPO(AD(EA(SI)<P>),AD,))  :(TSEI1)
275.      *    COMPONENT WITHOUT PARENTHESIZED FACTORS                         *
276.      CONA1   CONEX<1> = ADDEX(AS,AN,AC,AT,AD)
277.      *  ADDITIVE COMPONENTS OTHER THAN FIRST AND LAST                     **
278.      INVAP   P = P + 1
279.              FCT BAL . SI (ANY('+-') BAL) . FCT RPOS(0)  :F(CONLA)
280.              SI ANY('+-') $ AS FENCE (RNO | NULL) $ AN FENCE
281.      .       (CNS | NULL) $ AC FENCE ((BAL | NULL) . AT '/(' BAL . AD ')'
282.      .       RPOS(0) | (REM | NULL) $ AT NULL $ AD) =  ;  AN = EQ(AN,0) 1
283.              AD = DIFFER(AD) CONEX(AD)  ;  Q = 1  ;  SQEZ2(AT)
284.      *    EXPANSION OF PARENTHESIZED FACTORS                              *
285.      SUCN2   AC SCO . TMA '(' BAL . TMB ')' = TMA  :F(SUVR2)
286.              SI = MULPO(SI,CONEX(TMB),)  :(SUCN2)
287.      SUVR2   AT SVR . TMA '(' BAL . TMB (')@' ABORT | ')') = TMA  :F(TESI2)
288.              SI = MULPO(SI,CONEX(TMB),)  :(SUVR2)
289.      *    FIRST EXPANDED TERM                                             *
290.      TESI2   CONEX<P> = DIFFER(SI) EQ(EI(SI),1) ADDEX(SGNCO(AS(EA(SI)) AS),
291.      .       AN(EA(SI)) * AN,AC(EA(SI)) AC,AT(EA(SI)) AT,MULPO(AD(EA(SI)),AD,
292.      .       ))  :S(INVAP)
293.              CONEX<P> = DIFFER(SI) ADDEX(SGNCO(AS(EA(SI)<1>) AS),
294.      .       AN(EA(SI)<1>) * AN,AC(EA(SI)<1>) AC,AT(EA(SI)<1>) AT,
295.      .       MULPO(AD(EA(SI)<1>),AD,))  :F(CONA2)
296.      *    REMAINING EXPANDED TERMS                                        *
297.      TSEI2   P = LT(Q,EI(SI)) P + 1  :F(INVAP)  ;  Q = Q + 1
298.              CONEX<P> = ADDEX(SGNCO(AS(EA(SI)<Q>) AS),
299.      .       AN(EA(SI)<Q>) * AN,AC(EA(SI)<Q>) AC,AT(EA(SI)<Q>) AT,
```

```
300.      .       MULPO(AD(EA(SI)<Q>),AD,))  :(TSEI2)
301.      *    COMPONENT WITHOUT PARENTHESIZED FACTORS                          *
302.      CONA2   CONEX<P> = ADDEX(AS,AN,AC,AT,AD)  :(INVAP)
303.      *  LAST ADDITIVE COMPONENT                                            **
304.      CONLA   FCT ANY('+-') $ AS FENCE (RNO | NULL) $ AN FENCE
305.      .       (CNS | NULL) $ AC FENCE ((BAL | NULL) . AT '/(' BAL . AD ')'
306.      .       RPOS(0) | (REM | NULL) $ AT NULL $ AD)  ;  AN = EQ(AN,0) 1
307.              AD = DIFFER(AD) CONEX(AD)  ;  SI =  ;  Q = 1  ;  SQEZ2(AT)
308.      *    EXPANSION OF PARENTHESIZED FACTORS                               *
309.      SUCN3   AC SCO . TMA '(' BAL . TMB ')' = TMA  :F(SUVR3)
310.              SI = MULPC(SI,CONEX(TMB),)  :(SUCN3)
311.      SUVR3   AT SVR . TMA '(' BAL . TMB (')@' ABORT | ')') = TMA  :F(TESI3)
312.              SI = MULPC(SI,CONEX(TMB),)  :(SUVR3)
313.      *    FIRST EXPANDED TERM                                              *
314.      TESI3   CONEX<P> = DIFFER(SI) EQ(EI(SI),1) ADDEX(SGNCO(AS(EA(SI)) AS),
315.      .       AN(EA(SI)) * AN,AC(EA(SI)) AC,AT(EA(SI)) AT,MULPO(AD(EA(SI)),AD,
316.      .       ))  :S(CONRE)
317.              CONEX<P> = DIFFER(SI) ADDEX(SGNCO(AS(EA(SI)<1>) AS),
318.      .       AN(EA(SI)<1>) * AN,AC(EA(SI)<1>) AC,AT(EA(SI)<1>) AT,
319.      .       MULPO(AD(EA(SI)<1>),AD,))  :F(CONA3)
320.      *    REMAINING EXPANDED TERMS                                         *
321.      TSEI3   P = LT(Q,EI(SI)) P + 1  :F(CONRE)  ;  Q = Q + 1
322.              CONEX<P> = ADDEX(SGNCO(AS(EA(SI)<Q>) AS),
323.      .       AN(EA(SI)<Q>) * AN,AC(EA(SI)<Q>) AC,AT(EA(SI)<Q>) AT,
324.      .       MULPO(AD(EA(SI)<Q>),AD,))  :(TSEI3)
325.      *    COMPONENT WITHOUT PARENTHESIZED FACTORS                          *
326.      CONA3   CONEX<P> = ADDEX(AS,AN,AC,AT,AD)
327.      CONRE   CONEX = EXFUN(P,CONEX)  :(RETURN)
328.      *  FUNCTION WITH LESS THAN TWO ADDITIVE COMPONENTS                     **
329.      ONEEX   FCT (ANY('+-') | NULL) $ AS FENCE (RNO | NULL) $ AN FENCE
330.      .       (CNS | NULL) $ AC FENCE ((BAL | NULL) . AT '/(' BAL . AD ')'
331.      .       RPOS(0) | (REM | NULL) $ AT NULL $ AD)  ;  AN = EQ(AN,0) 1
332.              AS = IDENT(AS) '+'  ;  AD = DIFFER(AD) CONEX(AD)  ;  SQEZ2(AT)
333.      *    EXPANSION OF PARENTHESIZED FACTORS                               *
334.      SUCN4   AC SCO . TMA '(' BAL . TMB ')' = TMA  :F(SUVR4)
335.              SI = MULPO(SI,CONEX(TMB),)  :(SUCN4)
336.      SUVR4   AT SVR . TMA '(' BAL . TMB (')@' ABORT | ')') = TMA  :F(TESI4)
337.              SI = MULPO(SI,CONEX(TMB),)  :(SUVR4)
338.      *    FIRST EXPANDED TERM                                              *
339.      TESI4   CONEX<1> = DIFFER(SI) EQ(EI(SI),1) ADDEX(SGNCO(AS(EA(SI)) AS),
340.      .       AN(EA(SI)) * AN,AC(EA(SI)) AC,AT(EA(SI)) AT,MULPO(AD(EA(SI)),AD,
341.      .       ))  :S(ONCOR)
342.              CONEX<1> = DIFFER(SI) ADDEX(SGNCO(AS(EA(SI)<1>) AS),
343.      .       AN(EA(SI)<1>) * AN,AC(EA(SI)<1>) AC,AT(EA(SI)<1>) AT,
344.      .       MULPO(AD(EA(SI)<1>),AD,))  :F(CONA4)
345.      *    REMAINING EXPANDED TERMS                                         *
346.      TSEI4   P = LT(P,EI(SI)) P + 1  :F(ONCOR)
347.              CONEX<P> = ADDEX(SGNCO(AS(EA(SI)<P>) AS),
348.      .       AN(EA(SI)<P>) * AN,AC(EA(SI)<P>) AC,AT(EA(SI)<P>) AT,
349.      .       MULPO(AD(EA(SI)<P>),AD,))  :(TSEI4)
350.      *    COMPONENT WITHOUT PARENTHESIZED FACTORS                          *
351.      CONA4   CONEX = EXFUN(1,ADDEX(AS,AN,AC,AT,AD))  :(RETURN)
352.      ONCOR   CONEX = EXFUN(P,CONEX)  :(RETURN)
353.      ********************************************************************
354.      ****  DETERMINATION AND ORDERING OF AVAILABLE FUNCTIONS  ***********
355.      SQEZ2   Q = DIFFER(FCT) 0  :F(FRETURN)
356.      *  FUNCTIONS EXP, SIN, COS, LGN AND SQT                             **
357.      EXSQE   FCT TAB(Q) BREAK('ESCL') USF . TMA BAL . ARG ')' @Q  :S(SQARG)
358.              Q = 0  :(PASQE)
359.      SQARG   SQEZ2(ARG)
```

```
*     TEST FOR EXISTING FUNCTION TYPE                                    *
          P = 11
TEXTY     L = IDENT(CMP<P><1>,TMA) 2  :S(TEXAR)
          P = LT(P,CMP<0><2>) P + 1  :S(TEXTY)
          P = NE(CMP<0><2>,10) P + 1  :(NEWFU)
TEXAR     L = DIFFER(CMP<P><L>,ARG) L + 1  :F(EXSQE)
          CMP<P><L> = GT(L,CMP<P><0>) ARG  :F(TEXAR)
          CMP<P><0> = CMP<P><0> + 1  ;  ¬(DIFFER(TMA,'SIN(')
.         DIFFER(TMA,'COS('))  :F(EXSQE)  ;  NE(P,11)  :F(TRINE)
          CMP<P - 1><L> = ¬(DIFFER(CMP<P - 1><1>,'SIN(')
.         DIFFER(CMP<P - 1><1>,'COS(')) ARG  :F(TRINE)
          CMP<P - 1><0> = CMP<P - 1><0> + 1  :(EXSQE)
TRINE     CMP<P + 1><L> = ARG
          CMP<P + 1><0> = CMP<P + 1><0> + 1  :(EXSQE)
*     PROCEDURE FOR NEW FUNCTION TYPE                                    *
NEWFU     CMP<P> = TABLE(4,2)  ;  CMP<P><1> = TMA  ;  CMP<P><2> = ARG
          CMP<P><0> = 2  ;  P = P + 1
          CMP<P> = IDENT(TMA,'SIN(') TABLE(4,2)  :S(SYMEX)
          CMP<P> = IDENT(TMA,'COS(') TABLE(4,2)  :S(EXSYM)
          CMP<0><2> = CMP<0><2> + 1  :(EXSQE)
SYMEX     CMP<P><1> = 'COS('  :(ASGAR)
EXSYM     CMP<P><1> = 'SIN('
ASGAR     CMP<P><2> = ARG  ;  CMP<P><0> = 2
          CMP<0><2> = CMP<0><2> + 2  :(EXSQE)
*  FUNCTIONS ENCLOSED IN PARENTHESES AND RAISED TO A POWER              **
PASQE     FCT TAB(Q) SFU ('(' BAL . ARG ')') . TMA '@(' BAL ')' @Q
.         :F(RETURN)  ;  SQEZ2(ARG)
*     TEST FOR EXISTING FUNCTION TYPE AND PROCEDURE FOR NEW TYPE         *
          P = 31
TPATY     IDENT(CMP<P>,TMA)  :S(PASQE)
          P = LE(P,CMP<0><3>) P + 1  :S(TPATY)
          CMP<P> = TMA  ;  CMP<0><3> = CMP<0><3> + 1  :(PASQE)
*********************************************************************
****  MULTIPLICATION OF FUNCTIONS  ************************************
MULPO     MULPO = IDENT(TMA) TMB  :S(RETURN)
          MULPO = IDENT(TMB) TMA  :S(RETURN)
          A = GT(EI(TMA),1) 1  :S(ACITA)
          B = GT(EI(TMB),1) 1  :S(ACITB)
*  NEITHER TERM HAS ADDEX ARRAY                                         **
          MULPO = EXFUN(1,ADDEX(SGNCO(SGN AS(EA(TMA)) AS(EA(TMB))),
.         AN(EA(TMA)) * AN(EA(TMB)),AC(EA(TMA)) AC(EA(TMB)),
.         AT(EA(TMA)) AT(EA(TMB)),MULPO(AD(EA(TMA)),AD(EA(TMB)),)))
.         :(RETURN)
*  FIRST TERM HAS ADDEX ARRAY, SECOND IS TESTED                         **
ACITA     B = GT(EI(TMB),1) 1  :S(ACIBT)
          MULPO = EXFUN(EI(TMA),ARRAY(EI(TMA)))
CYFTA     EA(MULPO)<A> = ADDEX(SGNCO(SGN AS(EA(TMA)<A>) AS(EA(TMB))),
.         AN(EA(TMA)<A>) * AN(EA(TMB)),AC(EA(TMA)<A>) AC(EA(TMB)),
.         AT(EA(TMA)<A>) AT(EA(TMB)),MULPO(AD(EA(TMA)<A>),
.         AD(EA(TMB)),)  ;  A = LT(A,EI(TMA)) A + 1  :S(CYFTA)F(RETURN)
*  ONLY SECOND TERM HAS ADDEX ARRAY                                     **
ACITB     MULPO = EXFUN(EI(TMB),ARRAY(EI(TMB)))
CYFTB     EA(MULPO)<B> = ADDEX(SGNCO(SGN AS(EA(TMA)) AS(EA(TMB)<B>)),
.         AN(EA(TMA)) * AN(EA(TMB)<B>),AC(EA(TMA)) AC(EA(TMB)<B>),
.         AT(EA(TMA)) AT(EA(TMB)<B>),MULPO(AD(EA(TMA)),
.         AD(EA(TMB)<B>),)
          B = LT(B,EI(TMB)) B + 1  :S(CYFTB)F(RETURN)
*  BOTH TERMS HAVE ARRAYS                                               **
ACIBT     MULPO = EXFUN(EI(TMA) * EI(TMB),ARRAY(EI(TMA) * EI(TMB)))
CYFBT     EA(MULPO)<A + (B - 1) * EI(TMA)> = ADDEX(SGNCO(SGN
```

```
420.    .       AS(EA(TMA)<A>) AS(EA(TMB)<B>)),AN(EA(TMA)<A>) * AN(EA(TMB)<B>),
421.    .       AC(EA(TMA)<A>) AC(EA(TMB)<B>),AT(EA(TMA)<A>) AT(EA(TMB)<B>),
422.    .       MULPO(AD(EA(TMA)<A>),AD(EA(TMB)<B>),))
423.            A = LT(A,EI(TMA)) A + 1  :S(CYFBT)
424.            B = LT(B,EI(TMB)) B + 1  :F(RETURN)  ;  A = 1  :(CYFBT)
425.    ********************************************************************
426.    ****  SHORTENING OF SIGN STRINGS TO ONE TERM  ************************
427.    SGNCO   SGNCO '++' | '--' = '+'  :S(SGNCO)
428.            SGNCO '-+' | '+-' = '-'  :S(SGNCO)F(RETURN)
429.    ********************************************************************
430.    ****  COMMUTATION OF OPERATORS  **************************************
431.    OPCOM   OPCOM = ARRAY('0:' 2 * OI(OPA) * OI(OPB))
432.            OPCOM<0> = 2 * OI(OPA) * OI(OPB)
433.            A = 1  ;  B = 1  ;  C = 1  ;  SGN =
434.    PROPA   Q = NE(OA(OPA)<3,A>,0) 1  :F(BCINC)
435.            N = 2 ** (OA(OPA)<3,A> - 1)
436.            OA(OPA)<4,A> LEN(1) $ VAR (LEN(1) $ VR2 | NULL) (LEN(1) $ VR3 |
437.    .       NULL) (NULL | LEN(1) $ VR4 (LEN(1) $ VR5 | NULL) (LEN(1) $ VR6
438.    .       | NULL) (NULL | LEN(1) $ VR7 (LEN(1) $ VR8 | NULL) (LEN(1) $
439.    .       VR9 | NULL))) RPOS(0)
440.    CYHOB   OPCOM<C> = CYHAC(OPA,OPB,A,B,VAR,VR2,VR3,VR4,VR5,VR6,VR7,VR8,
441.    .       VR9)  ;  B = LT(B,OI(OPB)) B + 1  :F(CAINC)
442.            C = C + 2  ;  Q = 1  :(CYHOB)
443.    BCINC   A = LT(A,OI(OPA)) A + 1  :F(SETC2)
444.            C = C + 2 * OI(OPB)  :(PROPA)
445.    CAINC   A = LT(A,CI(OPA)) A + 1  :F(SETC2)  ;  B = 1
446.            C = C + 2  :(PROPA)
447.    SETC2   A = 1  ;  B = 1  ;  C = 2  ;  SGN = '-'
448.    PROPB   Q = NE(OA(OPB)<3,B>,0) 1  :F(ACINC)
449.            N = 2 ** (OA(OPB)<3,B> - 1)
450.            OA(OPB)<4,B> LEN(1) $ VAR (LEN(1) $ VR2 | NULL) (LEN(1) $ VR3 |
451.    .       NULL) (NULL | LEN(1) $ VR4 (LEN(1) $ VR5 | NULL) (LEN(1) $ VR6
452.    .       | NULL) (NULL | LEN(1) $ VR7 (LEN(1) $ VR8 | NULL) (LEN(1) $
453.    .       VR9 | NULL))) RPOS(0)
454.    CYHOA   OPCOM<C> = CYHAC(OPB,OPA,B,A,VAR,VR2,VR3,VR4,VR5,VR6,VR7,VR8,
455.    .       VR9)  ;  A = LT(A,OI(OPA)) A + 1  :F(CBINC)
456.            C = C + 2  ;  Q = 1  :(CYHOA)
457.    ACINC   B = LT(B,CI(OPB)) B + 1  :F(RETURN)
458.            C = C + 2 * OI(OPA)  :(PROPB)
459.    CBINC   B = LT(B,CI(OPB)) B + 1  :F(RETURN)  ;  A = 1
460.            C = C + 2  :(PROPB)
461.    ********************************************************************
462.    ****  CYCLIC DETERMINATION OF HALF A COMMUTATOR  *********************
463.    CYHAC   S = LT(S,OA(OPA)<3,A>) S + 1  :F(LSTVR)
464.    *  DETERMINATION FOR OPERATORS WITH ORDER GREATER THAN ONE            **
465.            DR = DERAD(OA(OPB)<1,B>)  :S(DETMA)
466.            Q = Q + 2 ** (OA(OPA)<3,A> - S)  :(NODTM)
467.    DETMA   TMA = OPDER(1,ARRAY('3,1'))  ;  OA(TMA)<1,1> = DR
468.            OA(TMA)<2,1> = OA(OPB)<2,B>  ;  OA(TMA)<3,1> = OA(OPB)<3,B>
469.            TMA = CYHAC(OPA,TMA,A,1,VR2,VR3,VR4,VR5,VR6,VR7,VR8,VR9,)
470.    NODTM   TMB = CYHAC(OPA,OPB,A,B,VR2,VR3,VR4,VR5,VR6,VR7,VR8,VR9,)
471.            S = S - 1
472.    *     ASSIGNMENTS WHEN ONE OF THE COMPONENTS IS NULL                   *
473.            CYHAC = IDENT(TMB) TMA  :S(RETURN)
474.            M = IDENT(TMA) 1  :F(AFCYH)
475.    TMBCY   OA(TMB)<2,M> = OA(TMB)<2,M> + 10 ** (IN<VAR>)
476.            OA(TMB)<3,M> = OA(TMB)<3,M> + 1
477.            M = LT(M,OI(TMB)) M + 1  :S(TMBCY)
478.            CYHAC = TMB  :(RETURN)
479.    *     ASSIGNMENT WHEN NEITHER COMPONENT IS NULL                        *
```

```
480.      AFCYH   CYHAC = OPDER(OI(TMA) + OI(TMB),ARRAY('3,' OI(TMA) + OI(TMB)))
481.              L = 1
482.      ASBAT   OA(CYHAC)<1,L> = OA(TMA)<1,L>
483.              OA(CYHAC)<2,L> = OA(TMA)<2,L>
484.              OA(CYHAC)<3,L> = OA(TMA)<3,L>
485.              L = LT(L,OI(TMA)) L + 1  :S(ASBAT)  ;  L = L + 1  ;  M = 1
486.      ASBBT   OA(CYHAC)<1,L> = OA(TMB)<1,M>
487.              OA(CYHAC)<2,L> = OA(TMB)<2,M> + 10 ** (IN<VAR>)
488.              OA(CYHAC)<3,L> = OA(TMB)<3,M> + 1
489.              L = LT(L,OI(CYHAC)) L + 1  :F(RETURN)  ;  M = M + 1  :(ASBBT)
490.      *  ASSIGNMENT FOR LAST OR ONLY DIFFERENTIATION VARIABLE                **
491.      LSTVR   DR = DERAD(OA(OPB)<1,B>)  :F(NOLST)
492.              Q = LT(Q,N) Q + 1  :F(LSTCY)
493.              CYHAC = OPDER(2,ARRAY('3,2'))
494.              OA(CYHAC)<1,1> = MULPO(OA(OPA)<1,A>,DR,SGN)
495.              OA(CYHAC)<2,1> = OA(OPB)<2,B>
496.              OA(CYHAC)<3,1> = OA(OPB)<3,B>
497.              OA(CYHAC)<1,2> = MULPO(OA(OPA)<1,A>,OA(OPB)<1,B>,SGN)
498.              OA(CYHAC)<2,2> = OA(OPB)<2,B> + 10 ** (IN<VAR>)
499.              OA(CYHAC)<3,2> = OA(OPB)<3,B> + 1  :(RETURN)
500.      LSTCY   CYHAC = OPDER(1,ARRAY('3,1'))
501.              OA(CYHAC)<1,1> = MULPO(OA(OPA)<1,A>,DR,SGN)
502.              OA(CYHAC)<2,1> = OA(OPB)<2,B>
503.              OA(CYHAC)<3,1> = OA(OPB)<3,B>  :(RETURN)
504.      NOLST   Q = LT(Q,N) Q + 1  :F(RETURN)
505.              CYHAC = OPDER(1,ARRAY('3,1'))
506.              OA(CYHAC)<1,1> = MULPO(OA(OPA)<1,A>,OA(OPB)<1,B>,SGN)
507.              OA(CYHAC)<2,1> = OA(OPB)<2,B> + 10 ** (IN<VAR>)
508.              OA(CYHAC)<3,1> = OA(OPB)<3,B> + 1  :(RETURN)
509.      ************************************************************************
510.      ****  PRODUCT AND ANTICOMMUTATION OF OPERATORS  ************************
511.      OPMUL   OPMUL = ARRAY('0:' OI(OPA) * OI(OPB) * SI)
512.              OPMUL<0> = OI(OPA) * OI(OPB) * SI
513.              A = 1  ;  B = 1  ;  C = 1
514.      ZEDIA   OPMUL<C> = EQ(OA(OPA)<3,A>,0) OPDER(1,ARRAY('3,1'))  :F(ATRAN)
515.      ELOPM   OA(OPMUL<C>)<1,1> = MULPO(OA(OPA)<1,A>,OA(OPB)<1,B>)
516.              OA(OPMUL<C>)<2,1> = OA(OPB)<2,B>
517.              OA(OPMUL<C>)<3,1> = OA(OPB)<3,B>
518.              C = C + SI  ;  B = LT(B,OI(OPB)) B + 1  :F(INCRA)
519.              OPMUL<C> = OPDER(1,ARRAY('3,1'))  :(ELOPM)
520.      ATRAN   OA(OPA)<4,A> LEN(1) $ VAR (LEN(1) $ VR2 | NULL) (LEN(1) $ VR3 |
521.      .       NULL) (NULL | LEN(1) $ VR4 (LEN(1) $ VR5 | NULL) (LEN(1) $ VR6
522.      .       | NULL) (NULL | LEN(1) $ VR7 (LEN(1) $ VR8 | NULL) (LEN(1) $
523.      .       VR9 | NULL))) RPOS(0)
524.      EICYM   OPMUL<C> = CYMUL(OPA,OPB,A,B,VAR,VR2,VR3,VR4,VR5,VR6,VR7,VR8,
525.      .       VR9)  ;  C = C + SI  ;  B = LT(B,OI(OPB)) B + 1  :S(EICYM)
526.      INCRA   A = LT(A,OI(OPA)) A + 1  :F(ANTCO)  ;  B = 1  :(ZEDIA)
527.      *  ANTICOMMUTATOR SEQUENCE                                            **
528.      ANTCO   A = EQ(SI,2) 1  :F(RETURN)  ;  B = 1  ;  C = 2
529.      ZEDIB   OPMUL<C> = EQ(OA(OPB)<3,B>,0) OPDER(1,ARRAY('3,1'))  :F(BTRAN)
530.      ELMOP   OA(OPMUL<C>)<1,1> = MULPO(OA(OPB)<1,B>,OA(OPA)<1,A>,)
531.              OA(OPMUL<C>)<2,1> = OA(OPA)<2,A>
532.              OA(OPMUL<C>)<3,1> = OA(OPA)<3,A>
533.              C = C + 2  ;  A = LT(A,OI(OPA)) A + 1  :F(INCRB)
534.              OPMUL<C> = OPDER(1,ARRAY('3,1'))  :(ELMOP)
535.      BTRAN   OA(OPB)<4,B> LEN(1) $ VAR (LEN(1) $ VR2 | NULL) (LEN(1) $ VR3 |
536.      .       NULL) (NULL | LEN(1) $ VR4 (LEN(1) $ VR5 | NULL) (LEN(1) $ VR6
537.      .       | NULL) (NULL | LEN(1) $ VR7 (LEN(1) $ VR8 | NULL) (LEN(1) $
538.      .       VR9 | NULL))) RPOS(0)
539.      EIMCY   OPMUL<C> = CYMUL(OPB,OPA,B,A,VAR,VR2,VR3,VR4,VR5,VR6,VR7,VR8,
```

```
540.      .       VR9)  ;  C = C + 2  ;  A = LT(A,OI(OPA)) A + 1  :S(EIMCY)
541.      INCRB   B = LT(B,OI(OPB)) B + 1  :F(RETURN)  ;  A = 1  :(ZEDIB)
542.      ************************************************************************
543.      ****  CYCLIC DETERMINATION OF OPERATOR PRODUCT  ************************
544.      CYMUL   S = LT(S,OA(OPA)<3,A>) S + 1  :F(MLSVR)
545.      *  DETERMINATION FOR OPERATORS WITH ORDER GREATER THAN ONE              **
546.              DR = DERAD(OA(OPB)<1,B>)  :F(ZEDER)
547.              TMA = OPDER(1,ARRAY('3,1'))  ;  OA(TMA)<1,1> = DR
548.              OA(TMA)<2,1> = OA(OPB)<2,B>  ;  OA(TMA)<3,1> = OA(OPB)<3,B>
549.              TMA = CYMUL(OPA,TMA,A,1,VR2,VR3,VR4,VR5,VR6,VR7,VR8,VR9,)
550.              TMB = CYMUL(OPA,OPB,A,B,VR2,VR3,VR4,VR5,VR6,VR7,VR8,VR9,)
551.      *     ASSIGNMENT WHEN NEITHER COMPONENT IS NULL                        *
552.              CYMUL = OPDER(OI(TMA) + OI(TMB),ARRAY('3,' OI(TMA) + OI(TMB)))
553.              S = S - 1  ;  L = 1
554.      TMASG   OA(CYMUL)<1,L> = OA(TMA)<1,L>
555.              OA(CYMUL)<2,L> = OA(TMA)<2,L>
556.              OA(CYMUL)<3,L> = OA(TMA)<3,L>
557.              L = LT(L,OI(TMA)) L + 1  :S(TMASG)  ;  L = L + 1  ;  M = 1
558.      TBASG   OA(CYMUL)<1,L> = OA(TMB)<1,M>
559.              OA(CYMUL)<2,L> = OA(TMB)<2,M> + 10 ** (IN<VAR>)
560.              OA(CYMUL)<3,L> = OA(TMB)<3,M> + 1
561.              L = LT(L,OI(CYMUL)) L + 1  :F(RETURN)  ;  M = M + 1  :(TBASG)
562.      *     ASSIGNMENT WHEN FIRST COMPONENT IS NULL                          *
563.      ZEDER   CYMUL = CYMUL(OPA,OPB,A,B,VR2,VR3,VR4,VR5,VR6,VR7,VR8,VR9,)
564.              S = S - 1  ;  M = 1
565.      NUFIT   OA(CYMUL)<2,M> = OA(CYMUL)<2,M> + 10 ** (IN<VAR>)
566.              OA(CYMUL)<3,M> = OA(CYMUL)<3,M> + 1
567.              M = LT(M,OI(CYMUL)) M + 1  :S(NUFIT)F(RETURN)
568.      *  ASSIGNMENT FOR LAST OR ONLY DIFFERENTIATION VARIABLE                **
569.      MLSVR   DR = DERAD(OA(OPB)<1,B>)  :F(NOMLS)
570.              CYMUL = OPDER(2,ARRAY('3,2'))
571.              OA(CYMUL)<1,1> = MULPO(OA(OPA)<1,A>,DR,)
572.              OA(CYMUL)<2,1> = OA(OPB)<2,B>
573.              OA(CYMUL)<3,1> = OA(OPB)<3,B>
574.              OA(CYMUL)<1,2> = MULPO(OA(OPA)<1,A>,OA(OPB)<1,B>,)
575.              OA(CYMUL)<2,2> = OA(OPB)<2,B> + 10 ** (IN<VAR>)
576.              OA(CYMUL)<3,2> = OA(OPB)<3,B> + 1  :(RETURN)
577.      NOMLS   CYMUL = OPDER(1,ARRAY('3,1'))
578.              OA(CYMUL)<1,1> = MULPO(OA(OPA)<1,A>,OA(OPB)<1,B>,)
579.              OA(CYMUL)<2,1> = OA(OPB)<2,B> + 10 ** (IN<VAR>)
580.              OA(CYMUL)<3,1> = OA(OPB)<3,B> + 1  :(RETURN)
581.      ************************************************************************
582.      ****  APPLICATION OF AN OPERATOR TO A FUNCTION  ************************
583.      OPAPL   OPAPL = ARRAY('0:' OI(OPA))  ;  OPAPL<0> = OI(OPA)  ;  A = 1
584.      NEAPL   OPAPL<A> = EQ(OA(OPA)<3,A>,0) OPDER(1,ARRAY('3,1'))  :F(SOMDE)
585.              OA(OPAPL<A>)<1,1> = MULPO(OA(OPA)<1,A>,FCT)  :(CHAVA)
586.      SOMDE   OA(OPA)<4,A> LEN(1) $ VAR (LEN(1) $ VR2 | NULL) (LEN(1) $ VR3 |
587.      .       NULL) (NULL | LEN(1) $ VR4 (LEN(1) $ VR5 | NULL) (LEN(1) $ VR6
588.      .       | NULL) (NULL | LEN(1) $ VR7 (LEN(1) $ VR8 | NULL) (LEN(1) $
589.      .       VR9 | NULL))) RPOS(0)
590.              OPAPL<A> = CYAPL(OPA,FCT,VAR,VR2,VR3,VR4,VR5,VR6,VR7,VR8,VR9)
591.              S = 1
592.      CHAVA   A = LT(A,OI(OPA)) A + 1  :S(NEAPL)F(RETURN)
593.      ************************************************************************
594.      ****  CYCLIC DETERMINATION OF OPERATOR APPLICATION TO FUNCTION  ********
595.      CYAPL   S = LT(S,OA(OPA)<3,A>) S + 1  :F(LORON)
596.              CYAPL = CYAPL(OPA,DERAD(FCT),VR2,VR3,VR4,VR5,VR6,VR7,VR8,VR9,)
597.      .       :S(RETURN)F(FRETURN)
598.      *  LAST OR ONLY DIFFERENTIATION VARIABLE                              **
599.      LORON   DR = DERAD(FCT)  :F(FRETURN)
```

```
600.            CYAPL = OPDER(1,ARRAY('3,1'))
601.            OA(CYAPL)<1,1> = MULPO(OA(OPA)<1,A>,DR,)  :(RETURN)
602.    ************************************************************************
603.    ****  DERIVATIVE  ******************************************************
604.    DERAD   ARG = VAR FUNST(FCT)  ;  ARG = GT(SIZE(ARG),60)  :S(CKSZF)
605.    *  CHECK FOR DERIVATIVE IN TABLE                                      **
606.            DERAD = DTA<ARG>  ;  DERAD = IDENT(DERAD) TDT<ARG>  :F(RETURN)
607.            DTA<ARG> = DIFFER(DERAD) DERAD  :S(RETURN)
608.    *  FUNCTIONS WITH ADDITIVE COMPONENTS                                 **
609.    CKSZF   GT(EI(FCT),1)  :F(DRAST)
610.            DERAD = DERAD(EXFUN(1,EA(FCT)<EI(FCT)>))  :F(DROIS)
611.            DR = DERAD(EXFUN(EI(FCT) - 1,EA(FCT)))  :S(DERCO)F(PURIA)
612.    DROIS   DERAD = DERAD(EXFUN(EI(FCT) - 1,EA(FCT)))  :S(PURIA)F(FRETURN)
613.    *  DERIVATIVE FOR FUNCTIONS WITH DIVISORS                             **
614.    DRAST   EA(FCT) = DIFFER(DATATYPE(EA(FCT)),'ADDEX') EA(FCT)<1>
615.            IDENT(AD(EA(FCT)))  :S(CVANU)
616.            DERAD = VARID(AD(EA(FCT))) MULPO(EXFUN(1,
617.    .       ADDEX('-' AS(EA(FCT)),AN(EA(FCT)),AC(EA(FCT)),
618.    .       AT(EA(FCT)),MULPO(AD(EA(FCT)),AD(EA(FCT))))),
619.    .       DERAD(AD(EA(FCT))))  :F(DINOF)
620.            DR = VARIN(AT(EA(FCT))) MULPO(EXFUN(1,ADDEX(
621.    .       AS(EA(FCT)),AN(EA(FCT)),AC(EA(FCT)),,AD(EA(FCT)))),
622.    .       DERAD(EXFUN(1,ADDEX(,1,,AT(EA(FCT)),))))  :S(DERCO)F(PURIA)
623.    *    DERIVATIVE WHEN DIVISOR IS NOT FUNCTION OF VARIABLE              *
624.    DINOF   DERAD = VARIN(AT(EA(FCT))) MULPO(EXFUN(1,ADDEX(
625.    .       AS(EA(FCT)),AN(EA(FCT)),AC(EA(FCT)),,AD(EA(FCT)))),
626.    .       DERAD(EXFUN(1,ADDEX(,1,,AT(EA(FCT)),))))  :S(PURIA)F(FRETURN)
627.    *  CHECK FOR VARIABLES IN FUNCTION (NULL DIVISOR)                     **
628.    CVANU   VARIN(AT(EA(FCT)))  :S(CNMUL)
629.            DERAD = IDENT(FCT,FAD) FAD  :S(PURIA)F(FRETURN)
630.    *  DERIVATIVE OF CONSTANT MULTIPLES OF VARIABLE                       **
631.    CNMUL   DERAD = IDENT(AT(EA(FCT)),VAR) EXFUN(1,ADDEX(AS(EA(FCT)),
632.    .       AN(EA(FCT)),AC(EA(FCT)),,))  :S(PURIA)
633.    *  FUNCTIONS WITH FACTORS WHICH ARE POWERS OF THE VARIABLE             **
634.            FCT = EXFUN(1,COPY(EA(FCT)))  ;  DR = ARRAY(4)
635.            AT(EA(FCT)) ((BAL | NULL) (NOTANY('QE(') | '(' BAL ')') | NULL)
636.    .       . DR<1> VAR ('@(' BAL . DR<4> ')' | NULL) = DR<1>  :F(PWRON)
637.    *    TEST FOR MULTIPLE OCCURRENCE OF FACTORS                          *
638.            AT(EA(FCT)) ((BAL | NULL) (NOTANY('QE(') | '(' BAL ')') | NULL)
639.    .       VAR  :S(CHTPV)
640.    *    DERIVATIVE OF PRODUCT OF VARIABLE AND OTHER FUNCTION             *
641.    DEVMU   DERAD = IDENT(DR<4>) FCT  :F(SINOP)
642.            DR = VARIN(AT(EA(FCT))) MULPO(DERAD(EXFUN(1,
643.    .       ADDEX(,1,,AT(EA(FCT)),))),EXFUN(1,ADDEX(AS(EA(FCT)),
644.    .       AN(EA(FCT)),AC(EA(FCT)),VAR,)))  :S(DERCO)F(PURIA)
645.    *    SEPARATION OF POWER INTO CONSTANT AND NUMERICAL COMPONENTS        *
646.    SINOP   DR<4> ((ANY('+-') | NULL) RNO) . DR<2> RPOS(0)  :S(STRIS)
647.            DR<4> BAL . DR<3> ((ANY('+-') | NULL) RNO) . DR<2> RPOS(0)
648.    .       :S(CFAEH)  ;  DR<3> = DR<4>  :(CFAEH)
649.    *    MULTIPLE OCCURRENCE OF FACTORS                                   *
650.    MOVAP   AT(EA(FCT)) ((BAL | NULL) (NOTANY('QE(') | '(' BAL ')') | NULL)
651.    .       . DR<1> VAR ('@(' BAL . DR<4> ')' | NULL $ DR<4>) = DR<1>
652.    .       :F(CFAEH)
653.    CHTPV   DR<1> = IDENT(DR<4>) 1  :S(TECMN)
654.    *    SEPARATION OF POWER INTO CONSTANT AND NUMERICAL COMPONENTS        *
655.            DR<4> ((ANY('+-') | NULL) RNO) . DR<1> RPOS(0)  :S(TECMN)
656.            DR<4> BAL . DR<4> ((ANY('+-') | NULL) RNO) . DR<1> RPOS(0)
657.            DR<3> = DR<3> SGNCO('+' DR<4>)
658.    TECMN   DR<2> = DR<2> + DR<1>  :(MOVAP)
659.    *    DERAD EVALUATION FAILURE DUE TO VARIABLE IN POWER                *
```

```
660.      CFAEH   OUTPUT = VARIN(DR<3>) ' COMPONENT ' DR<3> ' OF POWER OF '
661.      .       'VARIABLE ' VAR ' DEPENDS ON VARIABLE,'  :F(POWOK)
662.              OUTPUT = ' IN VIOLATION OF FORMAT RESTRICTIONS.'
663.              DERAD = FAD  :(PURIA)
664.      *     FORM TEST OF POWERS OF VARIABLE                                 *
665.      POWOK   NE(DR<2>,0)  :S(DEFEX)  ;  DIFFER(DR<3>)  :S(NULNO)
666.      *     EVALUATION FOR FACTOR WITH ZERO POWER                           *
667.              VARIN(AT(EA(FCT)))  :S(PWRON)F(FRETURN)
668.      *     POWERS WITH NONZERO NUMERICAL COMPONENTS                        *
669.      DEFEX   IDENT(DR<3>)  :F(NONUC)  ;  DR<4> = EQ(DR<2>,1)  :S(DEVMU)
670.      STRIS   DR<2> ('-' $ DR<3> | '+' | NULL) REM $ DR<4>
671.      *     DERIVATIVE FOR FUNCTIONS WITH NUMERICALLY POWERED FACTORS       *
672.              DERAD = NE(DR<2>,2) EXFUN(1,ADDEX(SGNCO(DR<3> AS(EA(FCT))),
673.      .       DR<4> * AN(EA(FCT)),AC(EA(FCT)),AT(EA(FCT))
674.      .       VAR '@(' DR<2> - 1 ')',))  :S(REONU)
675.              DERAD = EXFUN(1,ADDEX(AS(EA(FCT)),2 * AN(EA(FCT)),
676.      .       AC(EA(FCT)),AT(EA(FCT)) VAR,))
677.      REONU   DR = VARIN(AT(EA(FCT))) MULPO(DERAD(EXFUN(1,ADDEX(,1,,
678.      .       AT(EA(FCT)),,)),EXFUN(1,ADDEX(AS(EA(FCT)),AN(EA(FCT)),
679.      .       AC(EA(FCT)),VAR '@(' DR<2> ')',)))  :S(DERCO)F(PURIA)
680.      *     POWERS WITH ZERO NUMERICAL COMPONENTS                           *
681.      NULNO   DR<3> (ANY('+-') | NULL) . DR<2> FENCE (BAL ANY('+-')
682.      .       BAL RPOS(0) ABORT | REM $ DR<4>)  :S(PUCOP)
683.              DR<4> = '(' DR<3> ')'  ;  DR<2> =
684.      *     DERIVATIVE FOR FUNCTIONS WITH ZERO NUMERICAL COMPONENTS IN      *
685.      *     POWERS OF VARIABLE                                              *
686.      PUCOP   DERAD = EXFUN(1,ADDEX(SGNCO(DR<2> AS(EA(FCT))),
687.      .       AN(EA(FCT)),DR<4> AC(EA(FCT)),AT(EA(FCT)) VAR '@('
688.      .       DR<3> '-1)',))
689.              DR = VARIN(AT(EA(FCT))) MULPO(DERAD(EXFUN(1,ADDEX(,1,,
690.      .       AT(EA(FCT)),,)),EXFUN(1,ADDEX(AS(EA(FCT)),AN(EA(FCT)),
691.      .       AC(EA(FCT)),VAR '@(' DR<3> ')',)))  :S(DERCO)F(PURIA)
692.      *     DERIVATIVE FOR POWERS OF VARIABLE WITH NONZERO CONSTANT AND      *
693.      *     NUMERICAL COMPONENTS                                            *
694.      NONUC   DERAD = NE(DR<2>,1) EXFUN(1,ADDEX(AS(EA(FCT)),AN(EA(FCT)),
695.      .       '(' DR<3> SGNCO('+' DR<2>) ')' AC(EA(FCT)),AT(EA(FCT))
696.      .       VAR '@(' DR<3> SGNCO('+' DR<2> - 1) ')',))  :S(CKOTM)
697.              DERAD = EXFUN(1,ADDEX(AS(EA(FCT)),AN(EA(FCT)),
698.      .       '(' DR<3> '+1)' AC(EA(FCT)),AT(EA(FCT)) VAR '@(' DR<3> ')',))
699.      CKOTM   DR = VARIN(AT(EA(FCT))) MULPO(DERAD(EXFUN(1,ADDEX(,1,,
700.      .       AT(EA(FCT)),,)),EXFUN(1,ADDEX(AS(EA(FCT)),AN(EA(FCT)),
701.      .       AC(EA(FCT)),VAR '@(' DR<3> SGNCO('+' DR<2>) ')',)))
702.      .       :S(DERCO)F(PURIA)
703.      *  FUNCTIONS BEGINNING WITH FACTORS WHICH ARE POWERS                  **
704.      PWRON   AT(EA(FCT)) (ANY | '(' BAL ')' | USF BAL ')') . DR<1>
705.      .       '@(' BAL . DR<2> ')' =  :F(EXFUN)
706.      *     DERAD EVALUATION FAILURE DUE TO VARIABLE IN POWER               *
707.              OUTPUT = VARIN(DR<2>) ' POWER ' DR<2> ' OF FUNCTION ' DR<1>
708.      .       ' DEPENDS ON VARIABLE ' VAR ','  :F(PWROK)
709.              OUTPUT = ' IN VIOLATION OF FORMAT RESTRICTIONS.'
710.              DERAD = FAD  :(PURIA)
711.      *     TEST FOR VARIABLE IN FACTOR FUNCTION                            *
712.      PWROK   VARIN(DR<1>)  :F(PNOVA)
713.      *     SEPARATION OF POWER INTO CONSTANT AND NUMERICAL COMPONENTS      *
714.              DR<2> ('-' . DR<3> | '+' | NULL) RNO . DR<4> RPOS(0)  :S(NUMPO)
715.              DR<2> BAL . DR<3> ((ANY('+-') | NULL) RNO) . DR<4> RPOS(0)
716.      .       :S(NZEPO)
717.              DR<2> (ANY('+-') | NULL) . DR<3> FENCE (BAL ANY('+-')
718.      .       BAL RPOS(0) ABORT | REM $ DR<4>)  :S(ZENUC)
719.              DR<4> = '(' DR<2> ')'
```

```
720.      *     DERIVATIVE FOR FUNCTIONS WITH ZERO NUMERICAL COMPONENTS IN        *
721.      *     POWERS OF FACTOR FUNCTIONS                                        *
722.      ZENUC    DERAD = MULPO(DERAD(EXFUN(1,ADDEX(,1,,AT(EA(FCT)),))),
723.      .        EXFUN(1,ADDEX(DR<3> AS(EA(FCT)),DR<4> AC(EA(FCT)),DR<1> '@('
724.      .        DR<2> '-1)' AT(EA(FCT)),)))  :F(PNOVA)S(DREWA)
725.      *     DERIVATIVE FOR FACTOR FUNCTIONS WITH NUMERICAL POWERS             *
726.      NUMPO    DERAD = NE(DR<2>,2) MULPO(DERAD(EXFUN(1,ADDEX(,1,,DR<1>,))),
727.      .        EXFUN(1,ADDEX(DR<3> AS(EA(FCT)),DR<4> * AN(EA(FCT)),
728.      .        AC(EA(FCT)),DR<1> '@(' DR<2> - 1 ')' AT(EA(FCT)),)))  :S(DREWA)
729.               DERAD = MULPO(DERAD(EXFUN(1,ADDEX(,1,,DR<1>,))),EXFUN(1,
730.      .        ADDEX(AS(EA(FCT)),2 * AN(EA(FCT)),AC(EA(FCT)),
731.      .        DR<1> AT(EA(FCT)),)))  :F(PNOVA)S(DREWA)
732.      *     DERIVATIVE FOR FUNCTIONS BEGINNING WITH FACTORS WITH              *
733.      *     NONZERO CONSTANT AND NUMERICAL COMPONENTS IN POWERS               *
734.      NZEPO    DERAD = NE(DR<4>,1) MULPO(DERAD(EXFUN(1,ADDEX(,1,,DR<1>,))),
735.      .        EXFUN(1,ADDEX(AS(EA(FCT)),AN(EA(FCT)),'(' DR<2> ')'
736.      .        AC(EA(FCT)),DR<1> '@(' DR<3> SGNCO('+' DR<4> - 1) ')'
737.      .        AT(EA(FCT)),)))  :S(DREWA)
738.               DERAD = MULPO(DERAD(EXFUN(1,ADDEX(,1,,DR<1>,))),
739.      .        EXFUN(1,ADDEX(AS(EA(FCT)),AN(EA(FCT)),'(' DR<2> ')'
740.      .        AC(EA(FCT)),DR<1> '@(' DR<3> ')' AT(EA(FCT)),)))  :F(PNOVA)
741.      DREWA    DR = VARIN(AT(EA(FCT))) MULPO(EXFUN(1,ADDEX(
742.      .        AS(EA(FCT)),AN(EA(FCT)),AC(EA(FCT)),DR<1> '@(' DR<2> ')',)),
743.      .        DERAD(EXFUN(1,ADDEX(,1,,AT(EA(FCT)),))))  :S(DEPCO)F(PURIA)
744.      PNOVA    DERAD = VARIN(AT(EA(FCT))) MULPO(EXFUN(1,ADDEX(
745.      .        AS(EA(FCT)),AN(EA(FCT)),AC(EA(FCT)),DR<1> '@(' DR<2> ')',)),
746.      .        DERAD(EXFUN(1,ADDEX(,1,,AT(EA(FCT)),))))  :S(PURIA)F(FRETURN)
747.      *  FUNCTIONS BEGINNING WITH EXP, SQT, SIN, COS, AND LGN                 **
748.      EXFUN    AT(EA(FCT)) ('EXP' | 'SQT' | 'SIN' | 'COS' | 'LGN') $ DR<1>
749.      .        '(' BAL . DR<2> ')' =  :F(EFORM)S($(DR<1> 'FN'))
750.      *     DERIVATIVE OR COMPONENT, DEPENDING ON VARIABLE DEPENDENCE         *
751.      *     IN ARGUMENT                                                       *
752.      EXPFN    DERAD = VARIN(DR<2>) MULPO(DERAD(CONEX(DR<2>)),
753.      .        EXFUN(1,ADDEX(AS(EA(FCT)),AN(EA(FCT)),AC(EA(FCT)),
754.      .        'EXP(' DR<2> ')' AT(EA(FCT)),)))  :S(PROBO)F(OPROR)
755.      SQTFN    DERAD = VARIN(DR<2>) MULPO(DERAD(CONEX(DR<2>)),
756.      .        EXFUN(1,ADDEX(AS(EA(FCT)),0.5 * AN(EA(FCT)),AC(EA(FCT)),
757.      .        'SQT(' DR<2> ')@(-0.5)' AT(EA(FCT)),)))  :S(PROBO)F(OPROR)
758.      SINFN    DERAD = VARIN(DR<2>) MULPO(DERAD(CONEX(DR<2>)),
759.      .        EXFUN(1,ADDEX(AS(EA(FCT)),AN(EA(FCT)),AC(EA(FCT)),
760.      .        'COS(' DR<2> ')' AT(EA(FCT)),)))  :S(PROBO)F(OPROR)
761.      COSFN    DERAD = VARIN(DR<2>) MULPO(DERAD(CONEX(DR<2>)),
762.      .        EXFUN(1,ADDEX('-' AS(EA(FCT)),AN(EA(FCT)),AC(EA(FCT)),
763.      .        'SIN(' DR<2> ')' AT(EA(FCT)),)))  :S(PROBO)F(OPROR)
764.      LGNFN    DERAD = VARIN(DR<2>) MULPO(DERAD(CONEX(DR<2>)),
765.      .        EXFUN(1,ADDEX(AS(EA(FCT)),AN(EA(FCT)),AC(EA(FCT)),
766.      .        AT(EA(FCT)),CONEX(DR<2>))))  :F(OPROR)
767.      *     DERIVATIVE COMPONENT WHEN REST OF FUNCTION DEPENDS ON             *
768.      *     VARIABLE ALSO                                                     *
769.      PROBO    DR = VARIN(AT(EA(FCT))) MULPO(EXFUN(1,ADDEX(
770.      .        AS(EA(FCT)),AN(EA(FCT)),AC(EA(FCT)),DR<1> '(' DR<2> ')',)),
771.      .        DERAD(EXFUN(1,ADDEX(,1,,AT(EA(FCT)),))))  :S(DEPCO)F(PURIA)
772.      *     DERIVATIVE WHEN REST OF FUNCTION ONLY DEPENDS ON VARIABLE         *
773.      OPROR    DERAD = VARIN(AT(EA(FCT))) MULPO(EXFUN(1,ADDEX(
774.      .        AS(EA(FCT)),AN(EA(FCT)),AC(EA(FCT)),DR<1> '(' DR<2> ')',)),
775.      .        DERAD(EXFUN(1,ADDEX(,1,,AT(EA(FCT)),))))  :S(PURIA)F(FRETURN)
776.      *  CHECK FOR FORMAT ERRORS                                              **
777.      EFORM    AT(EA(FCT)) ANY('@/')  :S(MAFAI)
778.      *  FUNCTIONS WITH MULTIPLICATIVE COMPONENTS                             **
779.               AT(EA(FCT)) BAL . DR<1> BAL . AT(EA(FCT)) RPOS(0)  :F(FENPA)
```

```
780.            DERAD = VARIN(DR<1>) MULPO(DERAD(EXFUN(1,ADDEX(,1,,DR<1>,))),
781.    .       FCT)  :F(MULNO)
782.    MULYE   DR = VARIN(AT(EA(FCT))) MULPO(DERAD(EXFUN(1,
783.    .       ADDEX(,1,,AT(EA(FCT)),))),EXFUN(1,ADDEX(AS(EA(FCT)),
784.    .       AN(EA(FCT)),AC(EA(FCT)),DR<1>,)))  :S(DERCO)F(PURIA)
785.    MULNO   DERAD = VARIN(AT(EA(FCT))) MULPO(DERAD(EXFUN(1,
786.    .       ADDEX(,1,,AT(EA(FCT)),))),EXFUN(1,ADDEX(AS(EA(FCT)),
787.    .       AN(EA(FCT)),AC(EA(FCT)),DR<1>,)))  :S(PURIA)F(FRETURN)
788.    *  FUNCTIONS ENCLOSED IN PARENTHESES                                **
789.    FENPA   AT(EA(FCT)) '(' RTAB(1) . AT(EA(FCT)) ')'  :F(MAFAI)
790.            DERAD = MULPO(EXFUN(1,ADDEX(AS(EA(FCT)),AN(EA(FCT)),
791.    .       AC(EA(FCT)),,)),DERAD(CONEX(AT(EA(FCT)))),)  :S(PURIA)
792.    .       F(FRETURN)
793.    *  DERAD EVALUATION FAILURE DUE TO FORMAT ERROR                     **
794.    MAFAI   OUTPUT = ' FORMAT ERROR:  ATTEMPT TO DIFFERENTIATE '
795.    .       AT(EA(FCT)) ' W.R.T. ' VAR
796.            DERAD = FAD  :(PURIA)
797.    *  COMBINATION OF DERIVATIVE PARTS AND RETURN SEQUENCE              **
798.    DERCO   DERAD = EXFUN(EI(DERAD) + EI(DR),ARCHA(EA(DR),EA(DERAD),
799.    .       EI(DERAD) + EI(DR)))
800.    *     TEMPORARY TABLE ASSIGNMENT AND REGENERATION SEQUENCE           *
801.    PURIA   TDT<ARG> = DIFFER(ARG) DERAD  :F(RETURN)
802.            J = LT(J,K) J + 1  :S(RETURN)  ;  TDT = REGEN(TDT)  :(RETURN)
803.    ********************************************************************
804.    ****  FUNCTION IN STRING FORM FOR TABLE REFERENCING  ****************
805.    FUNST   P = DIFFER(DATATYPE(EA(FCT)),'ADDEX') 1  :S(DINFU)
806.            FUNST = IDENT(AD(EA(FCT))) AS(EA(FCT)) AN(EA(FCT)) AC(EA(FCT))
807.    .       AT(EA(FCT))  :S(RETURN)
808.            FUNST = AS(EA(FCT)) AN(EA(FCT)) AC(EA(FCT)) AT(EA(FCT)) '#'
809.    .       FUNST(AD(EA(FCT)))  :(RETURN)
810.    DINFU   FUNST = IDENT(AD(EA(FCT)<P>)) FUNST AS(EA(FCT)<P>)
811.    .       AN(EA(FCT)<P>) AC(EA(FCT)<P>) AT(EA(FCT)<P>)  :S(NELIS)
812.            FUNST = FUNST AS(EA(FCT)<P>) AN(EA(FCT)<P>) AC(EA(FCT)<P>)
813.    .       AT(EA(FCT)<P>) '#' FUNST(AD(EA(FCT)<P>)) '#'
814.    NELIS   GT(SIZE(FUNST),59)  :S(RETURN)
815.            P = LT(P,EI(FCT)) P + 1  :S(DINFU)F(RETURN)
816.    ********************************************************************
817.    ****  COMBINATION OF TWO ARRAYS INTO ONE  ***************************
818.    ARCHA   ARCHA = ARRAY(SI)  ;  P = 1  ;  IDENT(TMA)  :S(AREPU)
819.            ARCHA<1> = IDENT(DATATYPE(TMA),'ADDEX') TMA  :F(ARQFS)
820.            P = 2  :(AREPU)
821.    ARQFS   Q = 1
822.    ARASA   ARCHA<P> = TMA<Q>  :F(AREPU)
823.            P = LT(P,SI) P + 1  :F(RETURN)  ;  Q = Q + 1  :(ARASA)
824.    AREPU   IDENT(TMB)  :F(AREAD)  ;  EQ(P,1)  :S(ARCFL)F(AREPL)
825.    AREAD   ARCHA<P> = IDENT(DATATYPE(TMB),'ADDEX') TMB  :F(ARQSS)
826.    AREPL   EQ(P,SI)  :S(RETURN)F(ARCFL)
827.    ARQSS   Q = 1
828.    ARASB   ARCHA<P> = TMB<Q>  :F(ARCFL)  ;  P = LT(P,SI) P + 1  :F(RETURN)
829.            Q = Q + 1  :(ARASB)
830.    ARCFL   OUTPUT = ' INCORRECTLY LARGE VALUE FOR VARIABLE SI IN '
831.    .       'FUNCTION ARCHA.'  :(RETURN)
832.    ********************************************************************
833.    ****  CHECK FOR VARIABLES IN DIVISOR  *******************************
834.    VARID   P = DIFFER(DATATYPE(EA(FCT)),'ADDEX') 1  :S(MUTED)
835.            OUTPUT = DIFFER(AD(EA(FCT))) ' DERIVATIVE IS NOT EXECUTED '
836.    .       'CONSISTENTLY ON FUNCTION FORMATTED WITH DIVISOR OF DIVISOR;  '
837.    .       'SEE LATER INDICATION ALSO.'
838.            VARIN(AT(EA(FCT)))  :S(RETURN)F(FRETURN)
839.    MUTED   OUTPUT = DIFFER(AD(EA(FCT)<P>)) ' DERIVATIVE IS NOT EXECUTED '
```

```
840.      .       'CONSISTENTLY ON FUNCTION FORMATTED WITH DIVISOR OF DIVISOR;  '
841.      .       'SEE LATER INDICATION ALSO.'
842.              VARIN(AT(EA(FCT)<P>))  :S(RETURN)
843.              P = LT(P,EI(FCT)) P + 1  :S(MUTED)F(FRETURN)
844.      ************************************************************************
845.      ****  CHECK FOR VARIABLES IN STRING  ***********************************
846.      VARIN   &ANCHOR = 0  ;  FCT VAR  :S(XTVAR)
847.      CONST   &ANCHOR = 1  :(FRETURN)
848.      XTVAR   VAR ANY('XT')  :S($(VAR 'SECT'))
849.      YINVA   &ANCHOR = 1  :(RETURN)
850.      XSECT   FCT 'EXP(' =  :S(XSECT)  ;  FCT 'X'  :S(YINVA)F(CONST)
851.      TSECT   FCT 'SQT(' =  :S(TSECT)  ;  FCT 'T'  :S(YINVA)F(CONST)
852.      ************************************************************************
853.      ****  REGENERATION OF TEMPORARY DERIVATIVE TABLE  **********************
854.      REGEN   TBL = CONVERT(TBL,'ARRAY')
855.              REGEN = TABLE(40)  ;  P = K - 8
856.      PRESU   REGEN<TBL<P,1>> = TBL<P,2>
857.              P = LE(P,K) P + 1  :S(PRESU)  ;  J = 11
858.              K = 39  :(RETURN)
859.      ************************************************************************
860.      ****  ORDERING AND SIMPLIFICATION  *************************************
861.      SIMCO   SIMCO = ARRAY('0:9')  ;  P = 1  ;  N = 1  ;  C = 0
862.      TEZER   Q = DIFFER(OPER<P>) OA(OPER<P>)<3,1>  :S(ADDSQ)
863.              P = LT(P,OPER<0>) P + 1  :S(TEZER)F(FRETURN)
864.      *  DETERMINATION OF NUMBERS OF TERMS OF SAME ORDER                    **
865.      DONUT   N = 1
866.      CFONO   Q = DIFFER(OPER<P>) OA(OPER<P>)<3,1>  :S(ADDSQ)
867.              P = LT(P,OPER<0>) P + 1  :S(CFONO)F(DEFIN)
868.      QANON   Q = OA(OPER<P>)<3,N>
869.              OA(OPER<P>)<3,N> = GT(Q,9) 9  :F(ADDSQ)  ;  Q = 9
870.      ADDSQ   SIMCO<Q> = SIMCO<Q> + 1  ;  C = GT(Q,C) Q
871.              N = LT(N,CI(OPER<P>)) N + 1  :S(QANON)
872.              P = LT(P,OPER<0>) P + 1  :S(DONUT)
873.      DEFIN   P = 1
874.      *  CREATION OF ARRAYS INDEXED BY NUMBER OF DERIVATIVE COEFFICIENTS    **
875.      COANU   SIMCO<P> = NE(SIMCO<P>,0) ARRAY('2,0:' SIMCO<P>)
876.              P = LT(P,C) P + 1  :S(COANU)  ;  P = 1
877.              SIMCO<0> = DIFFER(SIMCO<0>) ARRAY('0:' SIMCO<0>)
878.      *  ASSIGNMENT OF OPDER COMPONENTS TO ARRAYS                           **
879.      ASOPA   N = 1
880.      MACFN   Q = DIFFER(OPER<P>) OA(OPER<P>)<3,1>  :S(MASTA)
881.              P = LT(P,OPER<0>) P + 1  :S(MACFN)F(ACFIN)
882.      QASAG   Q = OA(OPER<P>)<3,N>
883.      MASTA   EQ(Q,0)  :F(NOTRA)  ;  SIMCO<0><0> = SIMCO<0><0> + 1
884.              SIMCO<0><SIMCO<0><0>> = OA(OPER<P>)<1,N>  :(RESNA)
885.      NOTRA   SIMCO<Q><1,0> = SIMCO<Q><1,0> + 1
886.              SIMCO<Q><1,SIMCO<Q><1,0>> = OA(OPER<P>)<1,N>
887.              SIMCO<Q><2,SIMCO<Q><1,0>> = OA(OPER<P>)<2,N>
888.      RESNA   N = LT(N,CI(OPER<P>)) N + 1  :S(QASAG)
889.              P = LT(P,OPER<0>) P + 1  :S(ASOPA)
890.      *     NOW EXFUN COMPONENTS (OF OPDER) AND DC SET ORDERING INDEX ARE    *
891.      *     ASSIGNED TO ELEMENTS OF THE FORM SIMCO<M><1,N> AND SIMCO<M><2,N>, *
892.      *     RESPECTIVELY, WHERE M IS THE ORDER OF THE DC SET AND N IS AN     *
893.      *     INDEX FOR EACH COMPONENT.  THE NUMBER OF TERMS IN EACH M-INDEXED *
894.      *     ARRAY IS SIMCO<M><1,0>.  TERMS REFERENCED, E.G., BY K AND J, WITH *
895.      *     EQUAL VALUES FOR SIMCO<M><2,K> AND SIMCO<M><2,J>, MAY BE JOINED  *
896.      *     VIA COMBINATION OF THE EXFUN-ADDEX ARRAYS.                       *
897.      *  INDEXING OF TERMS LEXICALLY ORDERED BY ORDERING NUMBERS            **
898.      ACFIN   M = GT(C,0) C  :F(FAFNU)
899.      *     CREATION OF TABLE TO STORE ELEMENTS OF COMPONENTS WITH COMMON    *
```

```
900.      *     DC SET ORDER                                                    *
901.      EVABE   OPER = LT(1,SIMCO<M><1,0>) TABLE(10,10)  :S(GOSTA)
902.      *     FINAL ASSIGNMENT WHEN A CERTAIN ORDER CONTAINS ONLY ONE TERM    *
903.              SI = EQ(EI(SIMCO<M><1,1>),1) EA(SIMCO<M><1,1>)  :F(MOTHO)
904.              EA(SIMCO<M><1,1>) = ARRAY(1)  ;  EA(SIMCO<M><1,1>)<1> = SI
905.      MOTHO   SIMCO<M> = OPDER(1,SIMCO<M>)  :(MINCR)
906.      *     ASSIGNMENT TO TABLE ELEMENT REPRESENTING DERIVATIVE COEFFICIENT  *
907.      *     SET OF HIGHEST LEXICAL ORDERING NUMBER.  THE ARRAY STORES <1>,   *
908.      *     LEXICAL ORDERING NUMBER;  <2> & <3>, NUMBER AND IDENTITIES OF    *
909.      *     CORRESPONDING EXFUN (OR EXFUN-ARRAY) COMPONENTS.                 *
910.      GOSTA   OPER<1> = ARRAY(3)  ;  OPER<1><1> = SIMCO<M><2,1>
911.              OPER<1><2> = 1  ;  OPER<1><3> = TABLE(10,10)
912.              OPER<1><3><1> = 1  ;  P = 2
913.      UNQSQ   OPER<1><1> = GT(SIMCO<M><2,P>,OPER<1><1>) SIMCO<M><2,P>
914.      .       :F(UNPSQ)  ;  OPER<1><2> = 1  ;  OPER<1><3><1> = P  :(NEXP1)
915.      UNPSQ   OPER<1><2> = EQ(SIMCO<M><2,P>,OPER<1><1>) OPER<1><2> + 1
916.      .       :F(NEXP1)  ;  OPER<1><3><OPER<1><2>> = P
917.      NEXP1   P = LT(P,SIMCO<M><1,0>) P + 1  :S(UNQSQ)
918.              Q = LT(OPER<1><2>,SIMCO<M><1,0>) 2  :S(KTOHA)
919.      *     FINAL ASSIGNMENT WHEN ALL TERMS OF A CERTAIN ORDER HAVE THE      *
920.      *     SAME DERIVATIVE COEFFICIENT                                      *
921.              N = EI(SIMCO<M><1,1>)  ;  P = 2
922.      AWASD   N = N + EI(SIMCO<M><1,P>)
923.              P = LT(P,OPER<1><2>) P + 1  :S(AWASD)
924.              SI = OPDER(1,ARRAY('2,1'))
925.              OA(SI)<1,1> = EXFUN(N,ARRAY(N))
926.              OA(SI)<2,1> = SIMCO<M><2,1>
927.              P = 1  ;  N = 1
928.      NEWVN   EA(OA(SI)<1,1>)<P> = EQ(EI(SIMCO<M><1,N>),1) EA(SIMCO<M><1,N>)
929.      .       :S(QINFA)
930.              EA(OA(SI)<1,1>)<P> = EA(SIMCO<M><1,N>)<1>  :F(SDIMR)
931.              Q = 2  ;  P = P + 1
932.      NEXQ1   EA(OA(SI)<1,1>)<P> = EA(SIMCO<M><1,N>)<Q>  :F(NOTP1)
933.              Q = Q + 1  ;  P = P + 1  :(NEXQ1)
934.      NOTP1   N = N + 1  :(NEWVN)
935.      QINFA   P = P + 1  ;  N = N + 1  :(NEWVN)
936.      SDIMR   SIMCO<M> = SI  :(MINCR)
937.      *     ASSIGNMENTS TO REST OF TABLE ELEMENTS, EACH ELEMENT CORRESPONDING *
938.      *     TO DISTINCT DERIVATIVE COEFFICIENT SETS AND ORDERED BY LEXICAL   *
939.      *     ORDERING NUMBERS                                                 *
940.      KTOHA   OPER<0> = OPER<1><2>
941.      NEWQS   OPER<Q> = ARRAY(3)  ;  P = 1
942.      CFVAO   OPER<Q><1> = LT(SIMCO<M><2,P>,OPER<Q - 1><1>) SIMCO<M><2,P>
943.      .       :S(FIREL)  ;  P = P + 1  :(CFVAO)
944.      FIREL   OPER<Q><2> = 1  ;  OPER<Q><3> = TABLE(10,10)
945.              OPER<Q><3><1> = P  ;  P = LT(P,SIMCO<M><1,0>) P + 1  :F(QSAVE)
946.      ANQSQ   OPER<Q><1> = LT(SIMCO<M><2,P>,OPER<Q - 1><1>)
947.      .       GT(SIMCO<M><2,P>,OPER<Q><1>) SIMCO<M><2,P>  :F(ANPSQ)
948.              OPER<Q><2> = 1  ;  OPER<Q><3><1> = P  :(NEXP2)
949.      ANPSQ   OPER<Q><2> = EQ(SIMCO<M><2,P>,OPER<Q><1>) OPER<Q><2> + 1
950.      .       :F(NEXP2)  ;  OPER<Q><3><OPER<Q><2>> = P
951.      NEXP2   P = LT(P,SIMCO<M><1,0>) P + 1  :S(ANQSQ)
952.              OPER<0> = OPER<0> + OPER<Q><2>
953.              Q = LT(OPER<0>,SIMCO<M><1,0>) Q + 1  :S(NEWQS)
954.      QSAVE   SIMCO<M><2,0> = Q
955.      *     FINAL ASSIGNMENT FOR TERMS OF A COMMON ORDER                     *
956.              Q = 1
957.      QLTEQ   P = 1  ;  OPER<Q><1> = 0
958.      AWAIG   OPER<Q><1> = OPER<Q><1> + EI(SIMCO<M><1,OPER<Q><3><P>>)
959.              P = LT(P,OPER<Q><2>) P + 1  :S(AWAIG)
```

```
960.              Q = LT(Q,SIMCO<M><2,0>) Q + 1  :S(QLTEQ)
961.              SI = OPDER(SIMCO<M><2,0>,ARRAY('2,' SIMCO<M><2,0>))  ;  L = 1
962.      NEWVL   OA(SI)<2,L> = SIMCO<M><2,OPER<L><3><1>>
963.              OA(SI)<1,L> = EXFUN(OPER<L><1>,ARRAY(OPER<L><1>))
964.              P = 1  ;  N = 1
965.      NCYCP   EA(OA(SI)<1,L>)<P> = EQ(EI(SIMCO<M><1,OPER<L><3><N>>),1)
966.      .       EA(SIMCO<M><1,OPER<L><3><N>>)  :S(FAQIN)
967.              EA(OA(SI)<1,L>)<P> = EA(SIMCO<M><1,OPER<L><3><N>>)<1>  :F(MDILR)
968.              Q = 2  ;  P = P + 1
969.      NEXQ2   EA(OA(SI)<1,L>)<P> = EA(SIMCO<M><1,OPER<L><3><N>>)<Q>  :F(NOTP2)
970.              Q = Q + 1  ;  P = P + 1  :(NEXQ2)
971.      NOTP2   N = N + 1  :(NCYCP)
972.      FAQIN   P = P + 1  ;  N = N + 1  :(NCYCP)
973.      MDILR   L = LT(L,OI(SI)) L + 1  :S(NEWVL)  ;  SIMCO<M> = SI
974.      MINCR   M = GT(M,1) M - 1  :F(FAFNU)
975.              DIFFER(SIMCO<M>)  :S(EVABE)F(MINCR)
976.      *     FINAL ASSIGNMENT FOR RESULT COMPONENT WITH NULL DC SET                *
977.      FAFNU   N = DIFFER(SIMCO<0>) EI(SIMCO<0><1>)  :F(OROPO)
978.              P = GT(SIMCO<0><0>,1) 2  :F(SIDAO)
979.      ASFZE   N = N + EI(SIMCO<0><P>)
980.              P = LT(P,SIMCO<0><0>) P + 1  :S(ASFZE)
981.      SIDAO   SI = OPDER(1,ARRAY('2,1'))
982.              OA(SI)<1,1> = EXFUN(N,ARRAY(N))  ;  P = 1  ;  N = 1
983.      NADVA   EA(OA(SI)<1,1>)<P> = EQ(EI(SIMCO<0><N>),1) EA(SIMCO<0><N>)
984.      .       :S(PNBOA)  ;  EA(OA(SI)<1,1>)<P> = EA(SIMCO<0><N>)<1>  :F(ZEOST)
985.              Q = 2  ;  P = P + 1
986.      NEXQ3   EA(OA(SI)<1,1>)<P> = EA(SIMCO<0><N>)<Q>  :F(NOTP3)
987.              Q = Q + 1  ;  P = P + 1  :(NEXQ3)
988.      NOTP3   N = N + 1  :(NADVA)
989.      PNBOA   P = P + 1  ;  N = N + 1  :(NADVA)
990.      ZEOST   SIMCO<0> = SI
991.      *     RESULTANT OPERATOR IS IN AN ORDERED FORMAT IN WHICH EXFUN             *
992.      *     COMPONENTS K & J OF A COMMON ORDER M MAY BE REFEFENCED BY             *
993.      *     OA(SIMCO<M>)<1,K> AND OA(SIMCO<M>)<1,J>.  THUS SIMCO<M> =            *
994.      *     OPDER(Q,ARRAY(2,Q)).  THESE EXFUN COMPONENTS ARE COEFFICIENTS OF     *
995.      *     DI SETS OA(SIMCO<M>)<2,K> AND OA(SIMCO<M>)<2,J>, RESPECTIVELY.       *
996.      *     OPERATOR SIMCO MAY BE SIMPLIFIED AND SHORTENED BY COMBINATION OF     *
997.      *     COMMON ADDEX TERMS WHICH ARE ELEMENTS OF ARRAY                       *
998.      *     EA(OA(SIMCO<M>)<1,N>).  FOR EACH N, OA(SIMCO<M>)<1,N> =              *
999.      *     EXFUN(Q,ARRAY(Q)), EVEN FOR Q = 1.                                   *
1000.     *  DISPLAY OF ORDERED OPERATOR OR RESULT                                  **
1001.     OROPO   OUTPUT = ' TIME AND COUNT AT END OF ORDERING OF '
1002.     .       'RESULT:  ' TIME() ' MSEC, ' &STCOUNT ' STATEMENTS.'
1003.             OUTPUT =  ;  OUTPUT = ' ORDERED RESULT:'  ;  OUTPUT =
1004.             OUTPUT = OPOUT(SIMCO)  ;  OUTPUT =  ;  OUTPUT = ' TIME AND '
1005.     .       'COUNT AT END OF OUTPUT ROUTINE:  ' TIME() ' MSEC, ' &STCOUNT
1006.     .       ' STATEMENTS.'
1007.     *  CANNONICAL ORDERING, SIMPLIFICATION AND CANCELLATION                   **
1008.             M = C  ;  L = OI(SIMCO<M>)  ;  SI = 1  ;  DR = 0
1009.     *     SELECTION OF ADDEX COMPONENT                                         *
1010.     SELAC   OPER = EA(OA(SIMCO<M>)<1,L>)<SI>
1011.             I = DIFFER(AC(OPER))  :F(ORFUN)
1012.     *     TEST FOR PRESENCE OF IMAGINARY NUMBER INDICATOR "I"                  *
1013.             AC(OPER) (BAL | NULL) . TMA SPAN('I') . TMB = TMA  :F(NULCN)
1014.     INBYS   I = I + SIZE(TMB)     '
1015.     UNREM   I = GT(I,3) I - 4  :S(UNREM)
1016.             AC(OPER) (BAL | NULL) . TMA SPAN('I') . TMB = TMA  :S(INBYS)
1017.             I = EQ(I,0)  :S(NULCN)  ;  I = EQ(I,1) 'I'  :S(NULCN)
1018.             AS(OPER) = SGNCO('-' AS(OPER))  ;  I = EQ(I,2)  :S(NULCN)
1019.             I = 'I'
```

```
*    ORDERING OF CONSTANT FIELD OF ADDEX COMPONENT                          *
NULCN   P = DIFFER(AC(OPER)) 41  :F(IMONL)
        AC(OPER) = I CONCA(AC(OPER))  :S(ORFUN)
IMONL   AC(OPER) = I AC(OPER)
*    ORDERING OF FUNCTION FIELD OF ADDEX COMPONENT                          *
ORFUN   AT(OPER) = FUNCA(AT(OPER),CMP<0><1> + 1,)
        SI = LT(SI,EI(OA(SIMCO<M>)<1,L>)) SI + 1  :S(SELAC)
*    SIMPLIFICATION AND CANCELLATION WITHIN EXFUN COMPONENT                 *
        OA(SIMCO<M>)<1,L> = ADSUB(OA(SIMCO<M>)<1,L>)  :F(NULEX)
        DR = DR +'1  ;  FCF = GT(F,80) TABLE(30,80)  :F(EXONE)  ;  F = 1
EXONE   L = GT(L,1) L - 1  :F(TORCH)  ;  SI = 1  :(SELAC)
TORCH   M = GT(M,0) M - 1  :F(TIREC)
        L = DIFFER(SIMCO<M>) OI(SIMCO<M>)  :F(TORCH)
        SI = 1  :(SELAC)
NULEX   P = L
SHIFT   OA(SIMCO<M>)<1,P> = LT(P,OI(SIMCO<M>))
.       OA(SIMCO<M>)<1,P + 1>  :F(ILEQU)
        OA(SIMCO<M>)<2,P> = OA(SIMCO<M>)<2,P + 1>
        P = P + 1  :(SHIFT)
ILEQU   SIMCO<M> = EQ(P,1)  :S(CEQUM)
        OI(SIMCO<M>) = OI(SIMCO<M>) - 1
        L = GT(L,1) L - 1  :F(TORCH)  ;  SI = 1  :(SELAC)
CEQUM   C = EQ(M,C) C - 1  :F(TORCH)  ;  M = GT(M,0) M - 1  :F(FRETURN)
        L = DIFFER(SIMCO<M>) OI(SIMCO<M>)  :F(CEQUM)
        SI = 1  :(SELAC)
*  DISPLAY OF SIMPLIFIED RESULT                                             **
TIREC   CCF = GT(CC,50) TABLE(20,50)  :F(FUTA5)  ;  CC = 1
FUTA5   FCF = GT(F,80) TABLE(30,80)  :F(SIMOU)  ;  F = 1
SIMOU   OUTPUT = ' TIME AND COUNT AT END OF SIMPLIFICATION'
.       ' OF RESULT:  ' TIME() ' MSEC, ' &STCOUNT ' STATEMENTS.'
        OUTPUT =  ;  OUTPUT = ' SIMPLIFIED RESULT:'  ;  OUTPUT =
        OUTPUT = OPOUT(SIMCO)  ;  OUTPUT =  ;  OUTPUT =
        OUTPUT = DUPL('* -- *',20)  :(RETURN)
*************************************************************************
****  CONVERSION OF OPDER FOR OUTPUT  ***********************************
OPOUT   M = C  ;  Q = 1  ;  OUTPUT =  :(ADTMC)
DEVAM   M = GT(M,0) M - 1  :F(RETURN)
        Q = DIFFER(OPER<M>) 1  :F(DEVAM)  ;  OUTPUT =
*  CHECK FOR COMPONENT WITH ONLY ONE TERM                                   **
ADTMC   TBL = IDENT(DATATYPE(EA(OA(OPER<M>)<1,Q>)),'ADDEX')
.       EA(OA(OPER<M>)<1,Q>)  :S(CWOOT)
        TBL = EQ(EI(OA(OPER<M>)<1,Q>),1) EA(OA(OPER<M>)<1,Q>)<1>
.       :F(OASEQ)
CWOOT   AN(TBL) RTAB(1) . AN(TBL) '.'  ;  AS(TBL) = IDENT(AS(TBL),'+')
        TBL = IDENT(AD(TBL)) AS(TBL) AN(TBL) AC(TBL) AT(TBL)  :S(ONEML)
        TBL = AS(TBL) AN(TBL) AC(TBL) AT(TBL) '/(' DVOUT(AD(TBL)) ')'
        TBL (ANY('+-') | NULL) '1/'  :S(OVANU)
ONEML   TBL (ANY('+-') | NULL) . TMA '1' (RNO ABORT | (LEN(1) REM)
.       $ TMB) = TMA TMB
OVANU   OUTPUT = ' +< ' TBL ' >' ONTDC(OA(OPER<M>)<2,Q>,'D')  :(RENEQ)
*  SEQUENCE FOR COMPONENTS WITH MORE THAN ONE TERM                          **
OASEQ   P = 1
EVPCY   TBL = EA(OA(OPER<M>)<1,Q>)<P>  ;  AN(TBL) RTAB(1) . AN(TBL) '.'
        AS(TBL) = EQ(P,1) IDENT(AS(TBL),'+')
        TBL = IDENT(AD(TBL)) AS(TBL) AN(TBL) AC(TBL) AT(TBL)  :S(AONEM)
        TBL = AS(TBL) AN(TBL) AC(TBL) AT(TBL) '/(' DVOUT(AD(TBL)) ')'
        TBL (ANY('+-') | NULL) '1/'  :S(NEOIO)
AONEM   TBL (ANY('+-') | NULL) . TMA '1' (RNO ABORT | (LEN(1) REM)
.       $ TMB) = TMA TMB
*    COMPONENT IS DIVIDED INTO PIECES WHICH FIT ON A LINE                   *
```

```
1080.      NEOIO   OPOUT = LE(SIZE(OPOUT) + SIZE(TBL),90) OPOUT ' ' TBL  :F(NORFO)
1081.              OUTPUT = EQ(P,EI(OA(OPER<M>)<1,Q>)) ' +<' OPOUT ' >'
1082.      .       ONTDC(OA(OPER<M>)<2,Q>,'D')  :S(RENOP)  ;  P = P + 1  :(EVPCY)
1083.      NORFO   OUTPUT = GT(SIZE(OPOUT) + SIZE(TBL),105) ' +<' OPOUT  :F(MAFOL)
1084.              OUTPUT = EQ(P,EI(OA(OPER<M>)<1,Q>)) ' ' TBL ' >'
1085.      .       ONTDC(OA(OPER<M>)<2,Q>,'D')  :S(RENOP)
1086.              OPOUT = ' ' TBL  :(SEAOL)
1087.      MAFOL   OUTPUT = EQ(P,EI(OA(OPER<M>)<1,Q>)) ' +<' OPOUT ' ' TBL ' >'
1088.      .       ONTDC(OA(OPER<M>)<2,Q>,'D')  :S(RENOP)
1089.              OUTPUT = ' +<' OPOUT ' ' TBL  ;  OPOUT =
1090.      *    SECOND AND SUBSEQUENT LINES OF OUTPUT                          *
1091.      SEAOL   P = P + 1
1092.      SEVCY   TBL = EA(OA(OPER<M>)<1,Q>)<P>  ;  AN(TBL) RTAB(1) . AN(TBL) '.'
1093.              TBL = IDENT(AD(TBL)) AS(TBL) AN(TBL) AC(TBL) AT(TBL)  :S(SONEM)
1094.              TBL = AS(TBL) AN(TBL) AC(TBL) AT(TBL) '/(' DVOUT(AD(TBL)) ')'
1095.              TBL ANY('+-') '1/'  :S(XAMSZ)
1096.      SONEM   TBL ANY('+-') . TMA '1' (RNO ABORT | (LEN(1) REM) $ TMB) =
1097.      .       TMA TMB
1098.      XAMSZ   OPOUT = LE(SIZE(OPOUT) + SIZE(TBL),93) OPOUT ' ' TBL  :F(REOLO)
1099.              OUTPUT = EQ(P,EI(OA(OPER<M>)<1,Q>)) OPOUT ' >'
1100.      .       ONTDC(OA(OPER<M>)<2,Q>,'D')  :S(RENOP)F(SEAOL)
1101.      REOLO   OUTPUT = GT(SIZE(OPOUT) + SIZE(TBL),108) OPOUT  :F(HIROL)
1102.              OUTPUT = EQ(P,EI(OA(OPER<M>)<1,Q>)) ' ' TBL ' >'
1103.      .       ONTDC(OA(OPER<M>)<2,Q>,'D')  :S(RENOP)
1104.              OPOUT = ' ' TBL  :(SEAOL)
1105.      HIROL   OUTPUT = EQ(P,EI(OA(OPER<M>)<1,Q>)) OPOUT ' ' TBL ' >'
1106.      .       ONTDC(OA(OPER<M>)<2,Q>,'D')  :S(RENOP)
1107.              OUTPUT = OPOUT ' ' TBL  ;  OPOUT =  :(SEAOL)
1108.      RENOP   OPOUT =
1109.      RENEQ   OUTPUT =  ;  Q = LT(Q,OI(OPER<M>)) Q + 1  :F(DEVAM)S(ADTMC)
1110.      ********************************************************************
1111.      ****  STRING TRANSLATION OF DIVISOR ON OUTPUT  ***********************
1112.      DVOUT   P = DIFFER(DATATYPE(EA(FCT)),'ADDEX') 1  :S(PINCY)
1113.              AN(EA(FCT)) RTAB(1) . AN(EA(FCT)) '.'
1114.              AS(EA(FCT)) = IDENT(AS(EA(FCT)),'+')
1115.              DVOUT = AS(EA(FCT)) AN(EA(FCT)) AC(EA(FCT)) AT(EA(FCT))
1116.              OUTPUT = DIFFER(AD(EA(FCT))) ' * DIVISOR ' DVOUT(AD(EA(FCT)))
1117.      .       ' OF DIVISOR ' DVOUT '* DROPPED.'  :F(MBONE)
1118.              DVOUT = DVOUT '*'
1119.      MBONE   DVOUT (ANY('+-') | NULL) . TMA '1' (RNO ABORT | (LEN(1) REM)
1120.      .       $ TMB) = TMA TMB  :(RETURN)
1121.      PINCY   AN(EA(FCT)<P>) RTAB(1) . AN(EA(FCT)<P>) '.'
1122.              AS(EA(FCT)<1>) = EQ(P,1) IDENT(AS(EA(FCT)<1>),'+')
1123.              SI = AS(EA(FCT)<P>) AN(EA(FCT)<P>) AC(EA(FCT)<P>)
1124.      .       AT(EA(FCT)<P>)
1125.              SI (ANY('+-') | NULL) . TMA '1' (RNO ABORT | (LEN(1) REM)
1126.      .       $ TMB) = TMA TMB  ;  DVOUT = DVOUT SI
1127.              OUTPUT = DIFFER(AD(EA(FCT)<P>)) ' *' P '* DIVISOR '
1128.      .       DVOUT(AD(EA(FCT)<P>)) ' OF DIVISOR ' DVOUT '*' P '* DROPPED.'
1129.      .       :F(TENEP)  ;  DVOUT = DVOUT '*' P '*'
1130.      TENEP   P = LT(P,EI(FCT)) P + 1  :S(PINCY)F(RETURN)
1131.      ********************************************************************
1132.      ****  CONVERSION FROM ORDERING NUMBER TO DC STRING  ******************
1133.      ONTDC   ONTDC = GT(SI,0) ONTDC DR CMP<SIZE(SI)>  :F(RETURN)
1134.              SI = SI - 10 ** (SIZE(SI) - 1)  :(ONTDC)
1135.      ********************************************************************
1136.      ****  CONVERSION OF CONSTANT FIELD TO CANNONICAL FORM  ***************
1137.      CONCA   CONCA = PCT<FCT>  ;  CONCA = IDENT(CONCA) CCF<FCT>  :F(RETURN)
1138.              CONCA = IDENT(CONCA) FCT  :S(ROCOE)
1139.              PCT<FCT> = CONCA  :(RETURN)
```

```
*  TEST AND ASSIGNMENT FOR CONSTANT REFERENCED BY VALUE OF INDEX P     **
ROCOE   FCT (BAL | NULL) . TMA CMP<P> ('@(' BREAK(')') $ TMB ')' |
.       NULL $ TMB) = TMA  :S(CAFOR)
        P = LT(P,CMP<0><4>) P + 1  :S(ROCOE)
CNFAS   OUTPUT = ' CONSTANT EXPRESSION ' FCT ' CANNOT BE CONSISTENTLY '
.       'CONVERTED TO CANNONICAL FORM.'  ;  CCF<FCT> = FCT
        CC = CC + 1  :(FRETURN)
*  TEST FOR ADDITIONAL OCCURENCES OF CONSTANT REFERENCED BY P           **
CAFOR   FCT (BAL | NULL) CMP<P>  :F(EAZRE)  ;  Q = 0  :(TBSET)
OOCOC   FCT (BAL | NULL) . TMA CMP<P> ('@(' BREAK(')') $ TMB ')' |
.       NULL $ TMB) = TMA  :F(GAINF)
TBSET   TMB = IDENT(TMB) 1  ;  Q = TMB + Q  :(OOCOC)
EAZRE   Q = IDENT(TMB) 1  :S(GAINF)  ;  Q = TMB
GAINF   Q BREAK('.') . L '.' (NULL | OTN (NULL | OTN (NULL | OTN) |
.       '9' OTN) | '9' OTN (NULL | OTN) | '99' OTN) . M '9' SPAN('9')
.       RPOS(0)  :F(ELDCO)
        Q = IDENT(M) L + 1  :S(LSACT)  ;  Q = L '.' M + 1  :(LSACT)
ELDCO   Q RTAB(1) . Q '.'
LSACT   IDENT(FCT)  :F(NXCNS)
*  LAST CYCLE OF FUNCTION ACTION                                        **
        SI = EQ(Q,1) CMP<P>  :S(NOMOT)
        SI = EQ(Q,0)  :S(NOMOT)
        SI = CMP<P> '@(' Q ')'
NOMOT   CCF<CONCA> = SI  ;  CONCA = SI  ;  CC = CC + 1  :(RETURN)
*  CONSTANT INDEX IS SAVED, COMPARED WITH LIMIT AND INCREMENTED         **
NXCNS   N = P  ;  P = LT(P,CMP<0><4>) P + 1  :F(CNFAS)
*  CANNONICAL ORDERING CYCLE                                            **
        SI = EQ(Q,1) CMP<N> CONCA(FCT)  :S(NOMOT)
        SI = EQ(Q,0) CONCA(FCT)  :S(NOMOT)
        SI = CMP<N> '@(' Q ')' CONCA(FCT)  :S(NOMOT)
        CONCA = EQ(Q,1) CMP<N> FCT  :S(RETURN)
        CONCA = EQ(Q,0) FCT  :S(RETURN)
        CONCA = CMP<N> '@(' Q ')' FCT  :(RETURN)
*************************************************************************
****  CONVERSION OF FUNCTION FIELD TO CANNONICAL FORM  ******************
FUNCA   FUNCA = PFT<FCT>  ;  FUNCA = IDENT(FUNCA) FCF<FCT>  :F(RETURN)
        FUNCA = IDENT(FUNCA) FCT  :S(VAREF)
        PFT<FCT> = FUNCA  :(RETURN)
*  TEST FOR VARIABLES AND THEIR POWERS                                  **
VAREF   P = LT(P,10) GT(P,1) P - 1  :F(AVAFE)
VARFA   FACEX(CMP<P>)  :F(VAREF)  ;  TBL = CMP<P>  :(FLEVE)
*  TEST FOR AVAILABLE FUNCTIONS AND THEIR POWERS                        **
AVAFE   LT(P,10)  :F(CYFAV)  ;  P = LT(10,CMP<0><2>) 11  :F(PARFE)
        Q = 2  :(AVFAC)
CYFAV   LE(P,CMP<0><2>)  :F(PARFE)
SOMAV   Q = LT(Q,CMP<P><0>) Q + 1  :S(AVFAC)
        P = LT(P,CMP<0><2>) P + 1  :F(PARFE)  ;  Q = 2
AVFAC   FACEX(CMP<P><1> CMP<P><Q> ')')  :F(SOMAV)
        TBL = CMP<P><1> CMP<P><Q> ')'  :(FLEVE)
*  TEST FOR PARENTHESIZED FUNCTIONS AND THEIR POWERS                    **
PARFE   P = LT(P,30) LT(30,CMP<0><3>) 31  :S(PAFAC)
NEPAE   P = GT(P,30) LT(P,CMP<0><3>) P + 1  :F(FDONO)
PAFAC   FACEX(CMP<P>)  :F(NEPAE)  ;  TBL = CMP<P>  :(FLEVE)
*  FAILURE OF CONVERSION TO CANNONICAL FORM                             **
FDONO   OUTPUT = DIFFER(FUNCA) ' CONVERSION FAILURE OF FACTOR FUNCTION '
.       FCT  :F(NDUTZ)  ;  FCF<FCT> = FCT  ;  F = F + 1  :(FRETURN)
NDUTZ   OUTPUT = ' NULL CONVERSION FOR FUNCTION ' FCT '.'
        FCF<FCT> =  ;  F = F + 1  :(RETURN)
*  ASSIGNMENT OF FUNCTION VALUE                                         **
FLEVE   FUNCA = IDENT(FUNCA) IDENT(TMB) TBL  :S(FICAF)
```

```
1200.             FUNCA = IDENT(FUNCA) TBL '@(' TMB ')'  :S(FICAF)
1201.             FUNCA = IDENT(TMB) TBL FUNCA(FUNCA,P,Q)  :S(FICAF)
1202.             FUNCA = TBL '@(' TMB ')' FUNCA(FUNCA,P,Q)  :S(FICAF)
1203.             FUNCA = IDENT(TMB) TBL FUNCA  :S(FICAF)
1204.             FUNCA = TBL '@(' TMB ')' FUNCA
1205.       *  CREATION OF TABLE ELEMENTS                                    **
1206.       FICAF   FCF<FCT> = FUNCA  ;  F = F + 1  :(RETURN)
1207.       ******************************************************************
1208.       ****  EXTRACTION OF FUNCTIONAL FACTORS FROM FUNCTION STRINGS  **********
1209.       FACEX   FUNCA ((BAL | NULL) (NOTANY('QE(') | '(' BAL ')') | NULL)
1210.       .       . TMA TBL ('@(' BAL $ FACEX ')' | NULL $ FACEX) = TMA
1211.       .       :F(FRETURN)
1212.       IFANU   L = IDENT(FACEX) L + 1  :S(REFAC)
1213.               FACEX ((ANY('+-') | NULL) RNO) . N RPOS(0)  :S(LPLUQ)
1214.               FACEX BAL . FACEX ((ANY('+-') | NULL) RNO) . N RPOS(0)
1215.       .       :S(CONCO)  ;  TMB = TMB SGNCO('+' FACEX)  :(REFAC)
1216.       CONCO   TMB = TMB SGNCO('+' FACEX)
1217.       LPLUQ   L = L + N
1218.       REFAC   FUNCA ((BAL | NULL) (NOTANY('QE(') | '(' BAL ')') | NULL)
1219.       .       . TMA TBL ('@(' BAL $ FACEX ')' | NULL $ FACEX) = TMA
1220.       .       :S(IFANU)
1221.               TMB = IDENT(TMB) L  :F(POWIC)  ;  TMB = EQ(TMB,1)  :S(RETURN)
1222.               TMB = EQ(TMB,0)  :F(RETURN)S(FRETURN)
1223.       POWIC   TMB '+' =  ;  TMB = NE(L,0) TMB SGNCO('+' L)  :(RETURN)
1224.       ******************************************************************
1225.       ****  CANCELLATION AND ADDITION OF LIKE TERMS  ********************
1226.       ADSUB   ADSUB = EQ(EI(FCT),1) EXFUN(1,EA(FCT)<1>)  :S(RETURN)
1227.               Q = 1  ;  P = GT(EI(FCT),2) 2  :S(AMANY)
1228.       *  FUNCTION CONTAINING ONLY TWO ADDITIVE TERMS                   **
1229.               TMA = IDENT(AC(EA(FCT)<1>),AC(EA(FCT)<2>)) IDENT(AT(EA(FCT)<1>),
1230.       .       AT(EA(FCT)<2>)) IDENT(AD(EA(FCT)<1>),AD(EA(FCT)<2>))
1231.       .       EVAL(AS(EA(FCT)<1>) AN(EA(FCT)<1>) ' ' AS(EA(FCT)<2>) ' '
1232.       .       AN(EA(FCT)<2>))  :S(CANAD)  ;  ADSUB = FCT  :(RETURN)
1233.       CANAD   EQ(TMA,0)  :S(FRETURN)
1234.               TMA ANY('+-') $ AS(EA(FCT)<1>) REM $ AN(EA(FCT)<1>) |
1235.       .       REM $ AN(EA(FCT)<1>) ABORT  :S(TWATO)  ;  AS(EA(FCT)<1>) = '+'
1236.       TWATO   ADSUB = EXFUN(1,EA(FCT)<1>)  :(RETURN)
1237.       *  FUNCTION CONTAINING MANY ADDITIVE TERMS                       **
1238.       AMANY   TMA = AS(EA(FCT)<Q>) AN(EA(FCT)<Q>)
1239.       ATERA   TMA = IDENT(AC(EA(FCT)<Q>),AC(EA(FCT)<P>)) IDENT(AT(EA(FCT)<Q>),
1240.       .       AT(EA(FCT)<P>)) IDENT(AD(EA(FCT)<Q>),AD(EA(FCT)<P>)) TMA ' '
1241.       .       AS(EA(FCT)<P>) ' ' AN(EA(FCT)<P>)  :F(NOADD)
1242.               EA(FCT)<P> = LT(P,EI(FCT)) EA(FCT)<EI(FCT)>  :F(INOLA)
1243.               EA(FCT)<EI(FCT)> =  ;  EI(FCT) = EI(FCT) - 1  :(ATERA)
1244.       INOLA   EA(FCT)<EI(FCT)> =  ;  EI(FCT) = EI(FCT) - 1  :(EVATA)
1245.       NOADD   P = LT(P,EI(FCT)) P + 1  :S(ATERA)
1246.       EVATA   TMA = EVAL(TMA)
1247.               EA(FCT)<Q> = EQ(TMA,0) EA(FCT)<EI(FCT)>  :F(TANOZ)
1248.               EI(FCT) = GT(EI(FCT),1) EI(FCT) - 1  :F(FRETURN)
1249.               P = LT(Q,EI(FCT)) Q + 1  :S(AMANY)F(MARET)
1250.       TANOZ   TMA ANY('+-') $ AS(EA(FCT)<Q>) REM $ AN(EA(FCT)<Q>) |
1251.       .       REM $ AN(EA(FCT)<Q>) ABORT  :S(FITOS)  ;  AS(EA(FCT)<Q>) = '+'
1252.       FITOS   Q = LT(Q + 1,EI(FCT)) Q + 1  :F(MARET)  ;  P = Q + 1  :(AMANY)
1253.       MARET   ADSUB = EQ(EI(FCT),1) EXFUN(1,EA(FCT)<1>)  :S(RETURN)
1254.               ADSUB = FCT  :(RETURN)
1255.       ******************************************************************
1256.       ****  RE-INDEXING OF OPERATOR FOR RECOGNITION BY OPCOM AND OPMUL  ******
1257.       RESET   RESET = OPDER(DR,ARRAY('4,' DR))  ;  M = C  ;  P = 1
1258.       DIFTE   L = DIFFER(OPER<M>) 1  :F(MREDU)
1259.       ELASG   OA(RESET)<1,P> = OA(OPER<M>)<1,L>
```

```
1260.               OA(RESET)<2,P> = OA(OPER<M>)<2,L>
1261.               OA(RESET)<3,P> = M
1262.               OA(RESET)<4,P> = ONTOC(OA(OPER<M>)<2,L>,)
1263.               P = LT(P,OI(RESET)) P + 1   :F(RETURN)
1264.               L = LT(L,OI(OPER<M>)) L + 1   :S(ELASG)
1265.       MREDU   M = GT(M,0) M - 1   :S(DIFTE)F(RETURN)
1266.       ********************************************************************
1267.       ****  TERMINATION SEQUENCE  ******************************************
1268.       TERSQ   &DUMP = 0  ;  OUTPUT =
1269.       END
4000.       TERM
4001.       TERM
4002.       /*
```